
GENERACIÓN DE TOKENS PARA PROTEGER LOS DATOS DE TARJETAS BANCARIAS

TRABAJO TERMINAL No. 2017-B008

PRESENTAN

DANIEL AYALA ZAMORANO
LAURA NATALIA BORBOLLA PALACIOS
RICARDO QUEZADA FIGUEROA

DIRECTORA
DRA. SANDRA DÍAZ SANTIAGO

INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO



Índice

NOTACIÓN	XII
1. INTRODUCCIÓN	1
1.1. OBJETIVOS	4
1.1.1. OBJETIVO GENERAL	4
1.1.2. OBJETIVOS ESPECÍFICOS	5
1.2. JUSTIFICACIÓN	5
1.3. LA TOKENIZACIÓN EN OTROS CONTEXTOS	7
1.4. ORGANIZACIÓN DEL DOCUMENTO	8
2. MARCO TEÓRICO	9
2.1. INTRODUCCIÓN A LA CRIPTOGRAFÍA	10
2.1.1. CRIPTOANÁLISIS Y ATAQUES	11
2.1.2. CLASIFICACIÓN DE LA CRIPTOGRAFÍA	11
2.2. CIFRADOS POR BLOQUES	14
2.2.1. DEFINICIÓN	14
2.2.2. CRITERIOS PARA EVALUAR LOS CIFRADOS POR BLOQUE	15
2.2.3. REDES FEISTEL	15
2.2.4. DATA ENCRYPTION STANDARD (DES)	17
2.2.5. ADVANCED ENCRYPTION STANDARD (AES)	20
2.2.6. MODOS DE OPERACIÓN	24
2.3. FUNCIONES HASH	28
2.3.1. SECURE HASH ALGORITHM (SHA)	30

2.4. CIFRADOS QUE PRESERVAN EL FORMATO	30
2.4.1. CLASIFICACIÓN DE LOS CIFRADOS QUE PRESERVAN EL FORMATO	31
2.5. COMPOSICIÓN DEL NÚMERO DE UNA TARJETA	32
2.5.1. IDENTIFICADOR DEL EMISOR	32
2.5.2. NÚMERO DE CUENTA	33
2.5.3. DÍGITO VERIFICADOR	33
2.6. ESTÁNDARES DEL NIST Y PCI-SSC	33
I GENERACIÓN DE TOKENS	35
3. ALGORITMOS TOKENIZADORES	37
3.1. CLASIFICACIÓN DEL PCI SSC	38
3.2. ALGORITMOS TOKENIZADORES	38
3.2.1. ALGORITMO FFX	41
3.2.2. ALGORITMO BPS	43
3.2.3. ALGORITMO TKR2	50
3.2.4. ALGORITMO HÍBRIDO REVERSIBLE	52
3.2.5. TOKENIZACIÓN MEDIANTE GENERADOR DE NÚMEROS PSEUDOALEATORIOS	54
4. ANÁLISIS Y DISEÑO DE PROGRAMA TOKENIZADOR	57
4.1. REQUERIMIENTOS	58
4.1.1. IRREVERSIBLES	62
4.1.2. CRIPTOGRÁFICOS REVERSIBLES	62
4.1.3. NO CRIPTOGRÁFICOS REVERSIBLES	64
4.1.4. SEGUIMIENTO DE REQUERIMIENTOS DE TOKENS	69

4.2. DISEÑO DE PROGRAMA TOKENIZADOR	69
4.2.1. CLASES DE FFX	71
4.2.2. CLASES DE BPS	74
4.2.3. CLASES DE TKR	74
4.2.4. CLASES DE AHR	77
4.2.5. CLASES DE DRBG	79
4.2.6. ESTRUCTURA DE PRUEBAS DE UNIDADES	79
4.2.7. CODIFICADOR DE BINARIO A ASCII	83
5. IMPLEMENTACIÓN DE PROGRAMA TOKENIZADOR	85
5.1. TECNOLOGÍAS	86
5.1.1. DEPENDENCIAS DE DISTRIBUCIÓN	86
5.1.2. DEPENDENCIAS DE DESARROLLO	87
5.1.3. INSTRUCCIONES DE ENSAMBLADOR	88
5.2. PROGRAMA PARA GENERAR TOKENS	89
5.2.1. MÓDULO DE FFX	90
5.2.2. MÓDULO DE BPS	90
5.2.3. MÓDULO DE TKR	100
5.2.4. MÓDULO DE AHR	103
5.2.5. MÓDULO DE DRBG	103
5.3. RESULTADOS	109
5.3.1. RESULTADOS DE LAS PRUEBAS ESTADÍSTICAS A LOS DRBG	109
5.3.2. COMPARACIÓN DE DESEMPEÑO	114

II APPLICACIÓN WEB DE SISTEMA TOKENIZADOR	121
6. ANÁLISIS Y DISEÑO DE APLICACIÓN WEB	123
6.1. ANÁLISIS	124
6.1.1. REGLAS DE NEGOCIO	124
6.1.2. REQUERIMIENTOS	131
6.1.3. CASOS DE USO	136
6.1.4. INTERFACES DE USUARIO	152
6.1.5. CATÁLOGO DE MENSAJES	152
6.1.6. MODELO DE DATOS	165
6.1.7. SEGUIMIENTO DE REQUERIMIENTOS DE TOKENS	165
6.2. DISEÑO	168
6.2.1. DISEÑO DE LA BASE DE DATOS	168
6.2.2. VISTA ESTÁTICA	168
6.2.3. VISTA DINÁMICA	168
7. IMPLEMENTACIÓN DE APLICACIÓN WEB	175
7.1. TECNOLOGÍAS	176
7.1.1. DEPENDENCIAS DE DISTRIBUCIÓN	176
7.1.2. DEPENDENCIAS DE DESARROLLO	177
7.2. ASPECTOS RELEVANTES DE LA IMPLEMENTACIÓN	177
7.2.1. EJEMPLOS DEL PROGRAMA SERVIDOR	177
7.2.2. EJEMPLOS DEL PROGRAMA CLIENTE	183
7.3. RESULTADOS	188
7.3.1. DESEMPEÑO DE LA API	188

III CASO DE PRUEBA: TIENDA DE LIBROS EN LÍNEA	191
8. ANÁLISIS Y DISEÑO DE TIENDA EN LÍNEA	193
8.1. ANÁLISIS	194
8.1.1. CASOS DE USO	194
8.1.2. INTERFACES DE USUARIO	207
8.1.3. CATÁLOGO DE MENSAJES	207
8.2. DISEÑO	222
8.2.1. DISEÑO DE LA BASE DE DATOS	222
8.2.2. VISTA ESTÁTICA	222
8.2.3. VISTA DINÁMICA	222
9. IMPLEMENTACIÓN DE TIENDA EN LÍNEA	231
9.1. ASPECTOS RELEVANTES DE LA IMPLEMENTACIÓN	232
10. CONCLUSIONES	245
GLOSARIO	247
SIGLAS Y ACRÓNIMOS	256
BIBLIOGRAFÍA	263
LISTA DE FIGURAS	270
LISTA DE TABLAS	273

APÉNDICES	275
A. REQUERIMIENTOS PARA LAS PRIMITIVAS CRIPTOGRÁFICAS	277
B. GENERACIÓN DE BITS PSEUDOALEATORIOS	281
B.1. SEMILLAS	283
B.2. FUNCIONES	285
B.2.1. INSTANCIACIÓN	285
B.2.2. CAMBIO DE SEMILLA	285
B.2.3. GENERACIÓN	287
B.2.4. DESINSTANCIACIÓN	290
B.3. MECANISMOS BASADOS EN FUNCIONES HASH	290
B.4. MECANISMOS BASADOS EN CIFRADOS POR BLOQUE	291
B.5. GARANTÍAS	291
C. PRUEBAS ESTADÍSTICAS PARA GENERADORES DE NÚMEROS ALEATORIOS Y PSEUDOALEATORIOS	293
C.1. DEFINICIONES	294
C.1.1. GENERADORES ALEATORIOS	294
C.1.2. GENERADORES PSEUDOALEATORIOS	294
C.1.3. ALEATORIEDAD	295
C.1.4. IMPREDECIBILIDAD	295
C.1.5. PRUEBAS ESTADÍSTICAS	295
C.2. PRUEBAS	296
C.2.1. PRUEBA DE FRECUENCIA	297
C.2.2. PRUEBA DE FRECUENCIA EN UN BLOQUE	297

C.2.3. PRUEBA DE CARRERAS	297
C.2.4. PRUEBA DE LA CARRERA MÁS LARGA EN UN BLOQUE	297
C.2.5. PRUEBA DEL RANGO DE MATRIZ BINARIA	298
C.2.6. PRUEBA ESPECTRAL	298
C.2.7. PRUEBA DE COINCIDENCIA SIN SUPERPOSICIÓN	298
C.2.8. PRUEBA DE COINCIDENCIA CON SUPERPOSICIÓN	298
C.2.9. PRUEBA ESTADÍSTICA UNIVERSAL DE MAURER	299
C.2.10 PRUEBA DE COMPLEJIDAD LINEAL	299
C.2.11 PRUEBA SERIAL	299
C.2.12 PRUEBA DE ENTROPÍA APROXIMADA	299
C.2.13 PRUEBA DE SUMAS ACUMULATIVAS	300
C.2.14 PRUEBA DE EXCURSIONES ALEATORIAS	300
C.2.15 PRUEBA VARIANTE DE EXCURSIONES ALEATORIAS	300
D. GENERACIÓN DE LLAVES	301
D.1. CONCEPTOS CLAVE	302
D.1.1. MÉTODOS PARA LA GENERACIÓN DE LLAVES	302
D.1.2. DÓNDE GENERAR LAS LLAVES	302
D.1.3. FUERZA DE LA SEGURIDAD	302
D.1.4. USOS DE LAS SALIDAS DE LOS RANDOM BIT GENERATOR (RBG)	303
D.1.5. GENERACIÓN DE PARES DE LLAVES ASIMÉTRICAS	303
D.1.6. GENERACIÓN DE LLAVES SIMÉTRICAS	303
D.2. FUNCIONES PSEUDOALEATORIAS (PRF)	304
D.3. FUNCIONES DE DERIVACIÓN DE LLAVES (KDF)	304

D.4. MODOS DE ITERACIÓN	305
D.4.1. COUNTER MODE	305
D.4.2. FEEDBACK MODE	306
D.4.3. DOUBLE PIPELINE MODE	307
D.4.4. JERARQUÍA DE LLAVES	307
D.5. CONSIDERACIONES DE SEGURIDAD	307
D.5.1. FUERZA CRIPTOGRÁFICA	307
D.5.2. LA LONGITUD DE LA LLAVE DE ENTRADA	309
D.5.3. TRANSFORMACIÓN DE MATERIAL DE LLAVES A LLAVES CRIPTOGRÁFICAS.	309
D.5.4. CODIFICACIÓN DE LOS DATOS DE ENTRADA	309
D.5.5. SEPARACIÓN ENTRE LLAVES	309
D.5.6. ENLACE DE CONTEXTO	309
E. ADMINISTRACIÓN DE LLAVES	311
E.1. TIPOS DE LLAVES	312
E.2. USOS DE LLAVES	313
E.3. CRIPTOPERIODOS	313
E.4. ESTADOS DE LLAVES Y TRANSICIONES	316
F. PÁGINAS ESTÁTICAS	319
F.1. PÁGINA DE INICIO	320
F.1.1. ¿QUÉ ES LA TOKENIZACIÓN?	320
F.1.2. ¿POR QUÉ UTILIZAR UN SERVICIO DE TOKENIZACIÓN?	321
F.1.3. ¿POR QUÉ UTILIZAR ESTE SERVICIO DE TOKENIZACIÓN?	322
F.1.4. ¿QUÉ ES EL PCI SSC?	322

F.1.5. SOBRE LOS ALGORITMOS DE TOKENIZACIÓN	322
F.1.6. UNA COMPARACIÓN	324
F.2. PÁGINA DE DOCUMENTACIÓN	325
F.2.1. CÓMO UTILIZAR LA API WEB	325
G. SOBRE ESTE DOCUMENTO	331
G.1. DEPENDENCIAS	332
G.2. COMANDOS PROPIOS	336

Notación

A continuación se describe la notación que se utilizará a lo largo de este documento.

Símbolo	Descripción
K	Llave
pk	Llave pública
sk	Llave privada o secreta
k_i	i -ésima subllave
K_I	Llave de entrada o <i>key derivation key</i> de una función de derivación de llaves
K_O	Material de llaves obtenido de una función de derivación de llaves
IV	Vector de inicialización
E	Operación de cifrado
E_K	Operación de cifrado utilizando la llave K
D	Operación de descifrado
D_K	Operación de descifrado utilizando la llave K
h	Función hash
h_k	Función hash que utiliza una llave k para calcular el valor
M	Mensaje en claro
C	Mensaje cifrado
\mathbb{Z}_n	Conjunto de los números enteros módulo n
$\{0, 1\}^r$	Cadena de bits de longitud r
$\{0, 1\}^*$	Cadena de bits de longitud arbitraria
mód	Operación módulo
mcd	Máximo común divisor
\oplus	Operación <i>XOR</i>
\boxplus	Suma modular
\boxminus	Resta modular
\parallel	Operación de concatenación
$\{X\}$	Indicación de que el uso de X es opcional
$[X]$	El entero más pequeño que es mayor o igual que X
$[X]_2$	Representación binaria del número X
φ	Función ϕ de Euler
\emptyset	Conjunto vacío

Tabla 1: Notación

Es menester aclarar que un mensaje no consiste solo en letras y números; el *mensaje* se refiere al conjunto de datos que van a ser cifrados o descifrados.

Las palabras o frases CON ESTE ESTILO DE LETRA representan referencias a otras partes del documento: entradas al glosario, a la lista de siglas y acrónimos, referencias cruzadas. Si está leyendo esto en una versión digital, puede ocupar las referencias para navegar por el documento. En una versión digital, las referencias a la bibliografía, los números de página y los números de referencia a otras secciones también funcionan como enlaces para navegar por el documento.

Capítulo 1

Introducción

«In the beginning the Universe was created. This has made a lot of people very angry and has been widely regarded as a bad move.»

The Restaurant at the End of the Universe, DOUGLAS ADAMS.

A finales de los ochenta, el uso de las computadoras y el internet comenzó a popularizarse; compañías ya establecidas, como aerolíneas y tiendas departamentales, y comerciantes independientes vieron una oportunidad de expandirse y el comercio en línea comenzó a tomar fuerza; sin embargo, casi nadie previó el impacto y auge que iba a tener, por lo que la mayoría de los sitios no se encontraban preparados para los ataques y robos de información. Las principales emisoras de tarjetas (MasterCard y Visa) reportaron entre 1988 y 1998 pérdidas de 750 millones de dólares debidas a fraudes con tarjetas bancarias: el crecimiento del comercio electrónico aunado a sistemas débilmente protegidos dio lugar a un rápido crecimiento de los fraudes relacionados con tarjetas bancarias; de hecho, para el año 2001, según [1], se tuvieron pérdidas de 1.7 miles de millones de dólares y, para el siguiente año habían aumentado a 2.1 miles de millones de dólares.

Las compañías emisoras de tarjetas comenzaron a proponer soluciones y, a finales de 1999, Visa publicó un documento con una serie de recomendaciones de seguridad para quienes realizaban transacciones en línea llamado CARDHOLDER INFORMATION SECURITY PROGRAM (CISP), este programa es el primer precursor del estándar actual PAYMENT CARD INDUSTRY (PCI) DATA SECURITY STANDARD (DSS). Lamentablemente, las recomendaciones no estaban unificadas y había inconsistencias entre ellas, por lo que pocas compañías fueron capaces de satisfacer completamente alguna de las normas publicadas.

Fue hasta finales de 2004 cuando se publicó el primer estándar unificado, respaldado por las compañías emisoras de tarjetas más importantes: el PCI DSS 1.0, en el cuál se indica a los comercios cómo mantener los datos bancarios seguros mediante protocolos de seguridad, y se hizo obligatorio para todos aquellos que realizaran más de 20,000 transacciones anuales. Aunque cada vez más compañías comenzaron a seguirlo e invertir para satisfacerlo, de nuevo fueron pocas las que alcanzaron a cumplirlo completamente, pues tiene una gran cantidad de requerimientos (controles estrictos de acceso, monitoreo regular de las redes, mantener programas de vulnerabilidades y políticas de seguridad de información, etcétera) y las compañías tendían a subestimar los costos que implica seguir el estándar [2], [3].

A finales del 2006 ocurrió una de las primeras grandes violaciones de datos que puso en guardia a todos: hubo una intrusión en los servidores de la empresa americana TJX y piratas ciberneticos robaron información de tarjetas de crédito, débito y transacciones de 94 millones de clientes registrados en su sistema. Ataques como este han continuado a lo largo de los años: la cadena de supermercados Hannaford Bros. sufrió un embate en 2008 y se vieron comprometidas 4.2 millones de cuentas, Target fue atacado en 2013 y 40 millones de cuentas fueron afectadas, y, al año siguiente, arremetieron contra Home Depot y la información de 56 millones de usuarios fue robada.

Durante la primera década del siglo XXI, el enfoque que se tenía era salvaguardar la información sensible en todo el sistema; tómese por ejemplo el caso de una tienda en línea: el número de tarjeta queda registrado en el área de clientes, pues se puede asociar con un perfil y evita tener que estar ingresando continuamente toda la información de la tarjeta; también queda registrada en el área de ventas, pues queda asociada a una compra o transacción; la información sensible parece estar en todos lados (al menos, no

está concentrada) y tener que protegerla constantemente resulta muy costoso. Pasados unos años, surge la idea de cambiar completamente el paradigma que se había estado utilizando hasta ahora: ¿por qué intentar proteger la información sensible en todos los lados donde sea que se encuentre, cuando se puede cambiar esa información por un valor sustituto y solo proteger esa pequeña parte del sistema? Así que, a mediados del 2011, el PCI DSS publicó un documento cuyo título en inglés es «*PCI DSS tokenization guidelines*»; a su vez, varias compañías comenzaron a ofrecer soluciones de tokenización¹ que quitaban casi por completo el peso de cumplir con el PCI DSS a los comerciantes, pues ellas se encargan de generar los TOKENS y almacenar los datos sensibles; mientras que ellos, los comerciantes, se quedan solo con el TOKEN; así, si la seguridad de su sitio es violada, los datos siguen estando seguros, pues no se puede robar lo que no existe dentro del sistema.

Aunque en este trabajo se hace especial referencia a la tokenización de los números de tarjetas, o PERSONAL ACCOUNT NUMBER (PAN), este proceso sirve para proteger otros datos, como números de seguridad social, claves de registro, números de teléfono, etcétera. De hecho, la tokenización no está limitada al mundo digital y ha estado presente desde hace mucho tiempo, por ejemplo, los billetes y monedas, o las fichas en un casino: uno deposita dinero y obtiene su equivalente en fichas (este proceso es el de tokenización) para jugar; aunque las fichas (o TOKENS) están vigiladas, si alguien las roba, el casino no pierde tanto (siempre y cuando no sean cambiadas por dinero, o sea, realizar el proceso de detokenización), pues es una mera representación, y no el dinero mismo. Lo mismo sucede con los TOKENS digitales: sustituyen información valiosa, por valores que carecen de significado y cuya pérdida no representa un peligro inminente o una violación de la privacidad.

Todos los métodos para generar TOKENS que se presentan en este trabajo (y la gran mayoría de los métodos conocidos) competen enteramente a la criptografía. Entre otras cosas, la criptografía permite proteger información de terceros no autorizados, esto es, permite obtener confidencialidad. La idea básica es transformar un mensaje de forma que solo el destinatario sea capaz de hacer la transformación inversa y leer el mensaje original. Aunque con métodos un poco distintos a los actuales, la criptografía tiene una larga historia: el uso más antiguo del que se tiene noticia es en jeroglíficos egipcios de alrededor de 1900 a. C.; a partir de ahí, hay evidencias de su uso en muchas otras culturas antiguas [4]. La historia de la criptografía moderna está muy relacionada con la historia de la computación: en muchas ocasiones, la principal motivación para el desarrollo de máquinas más potentes fue la posibilidad de crear un método de cifrado infalible (o del método para romperlo); el ejemplo más claro de esto es el desarrollo de métodos para descifrar los mensajes de los nazis, durante la segunda guerra mundial, lo que permitió a Alan Turing sentar las bases de la computación actual [5]. Esta dinámica de juego, en la cual cada participante está siempre buscando un método que los adversarios no pueden romper, hace de la criptografía una ciencia en constante movimiento: una vez que se encuentra una nueva vulnerabilidad en algún método, se trabaja

¹El término *tokenización* es un anglicismo y, en este documento, se refiere a la acción y efecto de *tokenizar*; es decir, dado un valor, obtener un valor sustituto (TOKEN). La *detokenización* es el proceso inverso: dado el valor sustituto (TOKEN), obtener el valor original. Estos préstamos lingüísticos se utilizan de manera constante a lo largo de este reporte.

para corregirlo o para reemplazarlo.

Uno de los aspectos más importantes de las herramientas criptográficas modernas es que buscan probar, dentro de los límites posibles, la seguridad de un algoritmo. Es por esto que hoy en día no basta con describir un nuevo método y esperar que sea lo suficientemente fuerte, sino que la presentación de un método debe de ir acompañada de los argumentos (las pruebas) que garantizan su seguridad. Hacer esto no es tarea sencilla (si lo fuera, el juego ya estaría ganado para uno de los bandos) pues involucra hacer suposiciones sobre las capacidades de los adversarios y, basándose en esto, garantizar que los recursos necesarios para romper el método probado se encuentran muy por encima de las capacidades supuestas en un inicio. En este contexto, las capacidades de un adversario se determinan por su poder de cómputo: ¿cuántas computadoras tiene a su disposición?, ¿qué tan rápidas son?, etcétera.

Lo anterior nos lleva a otro principio muy importante de la criptografía moderna: la seguridad se encuentra en los datos, no en el método. La historia de la criptografía muestra que durante mucho tiempo, la seguridad de un método consistía totalmente en mantenerlo en secreto: mientras los adversarios no supieran cómo se estaba cifrando algo, los mensajes permanecerían seguros; sin embargo, si el método se filtraba, todos los mensajes cifrados con ese método se veían comprometidos. Este esquema presenta un serio problema, pues hace dependiente de un solo secreto, el método, la seguridad de todos los mensajes. Esto llevó a la introducción del concepto de la llave dentro de la criptografía: un valor que solo conocen las entidades autorizadas a leer el mensaje, sin el cuál no pueden descifrarlo; de esta forma la seguridad de un mensaje se basa tanto en el conocimiento del método, como en el conocimiento de la llave. En la mayoría de los métodos usados hoy en día, la seguridad se basa enteramente en la llave, dejando el método abierto al escrutinio de todos. Hacer esto presenta varias ventajas: una muestra de que el algoritmo está bien diseñado y es lo suficientemente fuerte es que no necesita mantenerse en secreto; cuando el algoritmo es público y es usado por muchas entidades (entre más mejor) son mayores las posibilidades de encontrar vulnerabilidades y el tiempo de respuesta de las correcciones es menor.

La presentación que se hace en este trabajo de los métodos de tokenización tiene un enfoque totalmente criptográfico. De esta forma se busca demostrar que la tokenización es una aplicación de la criptografía, no una alternativa a esta. Es por esto que todo el capítulo del marco teórico está dedicado a los mecanismos criptográficos que se relacionan con los métodos para generar TOKENS.

1.1. Objetivos

1.1.1. Objetivo general

Implementar algoritmos criptográficos y no criptográficos para generar TOKENS con el propósito de proveer confidencialidad a los datos de las tarjetas bancarias².

²Tomando en cuenta la clasificación propuesta por el PAYMENT CARD INDUSTRY (PCI) SECURITY STANDARD COUNCIL (SSC), descrita en la sección 3.1.

1.1.2. Objetivos específicos

1. Revisar diversos algoritmos para la generación de TOKENS.
2. Diseñar e implementar un servicio web que proporcione el servicio de generación de TOKENS de tarjetas bancarias a, al menos, una tienda en línea.
3. Implementar una tienda en línea que use el servicio de generación de TOKENS.

1.2. Justificación

Dado que es un tema relativamente nuevo, la desinformación y la desconfianza respecto a la generación de TOKENS es latente aún, pues, por ejemplo, aunque el PAYMENT CARD INDUSTRY (PCI) DATA SECURITY STANDARD (DSS) describe los requerimientos que debe cumplir un sistema tokenizador, no indica qué hacer para satisfacerlos. Muchas empresas se aprovechan de esta desinformación para promocionarse como sistema tokenizador sin ser claras sobre sus métodos. A continuación se muestran las principales deficiencias encontradas como resultado de una investigación comparativa de las principales soluciones que existen en el mercado.

Shift4 Esta compañía reclama el crédito de haber inventado el paradigma de la tokenización. Denuncia que otras compañías y el mismo PCI SECURITY STANDARD COUNCIL (SSC) desvirtuaron el término; por esta razón, la compañía se refiere a su propio método como *TrueTokenization*. Aunque mencionan en todas partes las supuestas ventajas de sus métodos, lo único que dicen con claridad sobre la generación de TOKENS es que se trata de valores aleatorios, únicos y alfanuméricos. Esta descripción no representa en modo alguno una garantía, pues podrían estar usando un método cuya seguridad no esté formalmente probada [6], [7].

Bluepay TokenShield En este servicio, a pesar de que se ofrecen dos formas distintas de tokenizar, nunca se aclara cómo es que funciona alguno de estos procesos. Además, en [8] hay una breve descripción del flujo de datos del servicio, donde se dice que con el TOKEN se recupera y descifra la información bancaria sensible, lo que genera confusión, pues no se sabe si la tokenización se hace por medio de cifrados que preservan el formato³, o por medio de bases de datos con tablas que contienen pares PERSONAL ACCOUNT NUMBER (PAN)-TOKEN.

Braintree Compañía que ofrece distintos SOFTWARE DEVELOPMENT KIT (SDK) (para dispositivos móviles y para desarrollo web) que permiten interactuar con sus propios servidores para procesar pagos con tarjetas de crédito. En [9] dan una definición de TOKEN que no es compatible con el PCI SSC: la única forma de generar TOKENS, según su definición, es por métodos aleatorios. Aún

³Tipo de cifrado que permite que los mensajes ya cifrados se vean parecidos a los originales. Este tema se aborda en la sección 2.4.

aceptando esa definición, no se ofrecen mayores explicaciones: los clientes no saben de qué forma se están generando los valores aleatorios.

Merchant Link La solución de tokenización de Merchant Link es llamada TransactionVault. Su página da tres puntos importantes respecto a la tokenización:

- Utilizan tecnología de la *siguiente generación*.
- Los TOKENS son asociados con una tarjeta y una transacción.
- Todos los TOKENS generados tienen una longitud de 16 dígitos y pasan los controles de validez de las tarjetas bancarias (por ejemplo, el algoritmo de Luhn⁴).

Aunque en el tercer punto informan al cliente que los TOKENS creados preservan el formato, no indican cuáles algoritmos son utilizados para generarlos; aunado a esto, el segundo punto tampoco aporta mucha información: sí, utilizan algoritmos reversibles⁵, pues los TOKENS quedan ligados a una tarjeta o a una transacción, pero siguen sin especificar sus métodos [10].

Jack Henry Card Processing Solution La solución de tokenización ofrecida por Jack Henry Banks indica que se encarga de sustituir el número de tarjeta con un TOKEN numérico que pasa los controles de validez de las tarjetas bancarias; sin embargo, no dice qué métodos utiliza para generar los TOKENS. Indica también que parte de su plataforma está integrada con los servicios de tokenización provistos por MASTERCARD DIGITAL ENABLEMENT SERVICE (MDES) y VISA TOKEN SERVICES (VTS); sin embargo, estos servicios tampoco indican cómo generan los tokens. Finalmente, no queda claro quién se encarga de generar los TOKENS, ¿lo hace MDES, VTS o Jack Henry? [11]-[13]

Securosis La publicación en [14] tiene por objeto esclarecer el papel de la tokenización. Explica de forma bastante clara las ventajas del paradigma y el flujo de datos entre tienda, sistema tokenizador y banco; sin embargo, al igual que las empresas anteriores, carece de una explicación sobre los propios TOKENS, y algunas de las referencias hacia estos solo consigue confundir. Por ejemplo, presentan a la tokenización como una alternativa a la criptografía, hablando de los TOKENS como valores aleatorios que nada tienen que ver con criptografía.

Muchas de las soluciones del mercado intentan argumentar que un TOKEN es un valor aleatorio y, por tanto, nada tiene que ver con criptografía; al parecer esta estrategia publicitaria tuvo mucho éxito, pues uno de los mensajes más comunes es que la criptografía es cosa del pasado, la tokenización permite sustituirla. Mientras esto puede ser cierto para los posibles clientes de un sistema tokenizador (tiendas que procesan pagos en línea), es una mentira manifiesta cuando se considera a todos los participantes.

La mayoría de las soluciones tratan a sus métodos como secretos de compañía: no explican la manera en que generan los TOKENS, por lo que esperan que el trato entre cliente y proveedor esté basado en la

⁴Este algoritmo es descrito en 2.10.

⁵Algoritmos que, dado un TOKEN, pueden regresar al PAN correspondiente. Esta clasificación es explicada en 3.1.

confianza que tiene el primero por el segundo. El modelo basado en la confianza es común al funcionamiento de casi todas las transacciones monetarias hechas por internet: dos partes que quieran realizar una transacción necesitan de un tercero (la institución financiera) para llevarla a cabo. Un estado ideal eliminaría la necesidad de la tercera parte (es lo que intentan hacer CRIPTOMONEDAS como *bitcoin* [15]), sin embargo, mientras este estado no se alcance, el modelo basado en la confianza es un mal necesario, y como tal, debe ser reducido al máximo: un proveedor del servicio de tokenización debería ser muy claro sobre lo que hace para tokenizar, permitiendo que sus clientes basen en esto su decisión.

La desinformación que hay alrededor del tema de la tokenización es la principal justificación para este trabajo: se busca implementar algoritmos públicos y presentar los resultados, permitiendo a los clientes de sistemas tokenizadores entender qué hay dentro de la caja negra. Parte de este combate a la desinformación consiste en dejar bien clara la relación entre la tokenización y la criptografía: la tokenización es una aplicación de la criptografía, por lo que los métodos de tokenización deben ser analizados con la misma formalidad con la que se estudian todas las demás formas de cifrado.

1.3. La tokenización en otros contextos

Dado que el sustantivo TOKEN y el término *tokenización* serán utilizados una y otra vez a lo largo del presente documento, es imperativo definirlos claramente. Actualmente, el vocablo TOKEN es ampliamente utilizado y su significado cambia dependiendo del contexto en el que se encuentra; a continuación se enlistan una serie de definiciones que exemplifican lo cambiante que puede ser su definición:

En el procesamiento de lenguaje natural, la tokenización es el proceso de segmentar un texto en unidades lingüísticas, tales como palabras, signos de puntuación, números y símbolos; regularmente, un TOKEN es una palabra lingüísticamente significativa [16].

En compiladores, la tokenización (o análisis léxico) se refiere al proceso de convertir una secuencia de caracteres a una secuencia de TOKENS (cadenas con significados) [17].

En el contexto social, se le llama tokenismo a la práctica de hacer solo un intento simbólico de ser inclusivo con un colectivo discriminado para aparentar diversidad (racial, sexual, etc.); quienes pertenecen a la minoría son conocidos como TOKENS [18].

En el contexto bancario, se le denomina TOKEN a un pequeño aparato encargado de generar llaves para poder establecer una conexión con sus servidores. Es usado como método de AUTENTICACIÓN MULTIFACTOR.

En el contexto económico, se le denomina TOKEN a una pieza metálica o plástica que puede ser utilizada en vez del dinero; por ejemplo, las fichas que se utilizan en los casinos.

Probablemente una de las fuentes de confusión más grandes viene del contexto de las CRIPTOMONEDAS,

pues el auge y la presencia que han tenido en los medios los últimos años las han relacionado intrínsecamente con los TOKENS. Aunado a esto, dentro de este contexto, un TOKEN adquiere varios significados: puede utilizarse como sinónimo de CRIPTOMONEDA (*Ana tiene 42 tokens de Bitcoin*) o puede hacer referencia a la cadena de números y letras que identifican una transacción.

Después de identificar qué no significa un TOKEN, a continuación se provee una definición de lo que es un TOKEN en el contexto de la criptografía, particularmente, en el campo de los algoritmos tokenizadores: un TOKEN es un valor representativo de un dato sensible (p. ej. un número de seguridad social, un número de tarjeta) que carece de una relación directa con el dato sensible en cuestión; dicho de otra manera, partiendo del TOKEN, es imposible llegar al dato sensible sin los medios adecuados. El proceso mediante el cual se genera el TOKEN, es conocido como *tokenización* y esta hace uso de algoritmos tokenizadores; en el caso de este trabajo, los TOKENS resultantes de este proceso de tokenización son cadenas que *parecen* números de tarjetas bancarias pero que no lo son.

1.4. Organización del documento

El capítulo 2 provee un conjunto de resúmenes sobre los temas necesarios para la comprensión de este trabajo; adentrándose en la criptografía, sus objetivos y los métodos que emplea para la protección de información, tales como los cifrados por bloque y las funciones hash.

En el capítulo 3 se describen algunos de los estándares pertinentes a la generación de TOKENS, así como las descripciones de los algoritmos existentes para realizar esta labor.

El capítulo 4 muestra el análisis técnico para la implementación del programa tokenizador; abordando sus requerimientos y el diseño del propio programa.

El capítulo 5 contiene los fragmentos de código más importantes para entender el funcionamiento tanto de los algoritmos de tokenización como el programa de generación de TOKENS; también presenta los resultados de las comparaciones de desempeño entre los distintos algoritmos tokenizadores.

Por último, en el capítulo 10 se concluyen los resultados del trabajo, hablando sobre los avances que se hicieron y el trabajo que aún falta por hacerse.

Capítulo 2

Marco teórico

«Desvarío laborioso y empobrecedor el de componer vastos libros; el de explayar en quinientas páginas una idea cuya perfecta exposición oral cabe en pocos minutos. Mejor procedimiento es simular que esos libros ya existen y ofrecer un resumen, un comentario.»

El jardín de senderos que se bifurcan,
JORGE LUIS BORGES.

El marco teórico contiene la mayoría de los conceptos utilizados para la implementación de los algoritmos generadores de TOKENS. Comienza con la introducción a la criptografía, sus objetivos, ataques y tipos. Posteriormente, se tiene un resumen de los cifrados por bloque, haciendo énfasis en el funcionamiento de las redes Feistel y el cifrado ADVANCED ENCRYPTION STANDARD (AES) y los modos de operación ELECTRONIC CODEBOOK (ECB), CIPHER-BLOCK CHAINING (CBC) y COUNTER MODE (CTR); después, se agregan los cifrados que preservan el formato y finaliza con una descripción sobre la composición de un número de tarjeta y los estándares del NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY (NIST) y PAYMENT CARD INDUSTRY (PCI) SECURITY STANDARD COUNCIL (SSC) que se utilizaron en este trabajo. Cada sección indica al inicio las fuentes bibliográficas que fueron consultadas en caso de que el lector desee profundizar en algún tema tratado.

2.1. Introducción a la criptografía

La información presentada a continuación puede consultarse con más profundidad en las siguientes referencias [19], [20].

La palabra criptografía proviene de las etimologías griegas *Kriptos* (ocultar) y *Graphos* (escritura), y es definida por la REAL ACADEMIA ESPAÑOLA (RAE) como «el arte de escribir con clave secreta o de un modo enigmático». De manera más formal se puede definir a la criptografía como la ciencia encargada de estudiar y diseñar por medio de técnicas matemáticas, métodos y modelos capaces de resolver problemas en la seguridad de la información, como la confidencialidad de esta, su integridad y la autenticación de su origen.

La criptografía tiene como finalidad proveer los siguientes cuatro servicios:

Confidencialidad Es el servicio encargado de mantener legible la información solo a aquellos que estén autorizados a visualizarla.

Integridad Este servicio se encarga de evitar la alteración de la información de forma no autorizada, esto incluye la inserción, sustitución y eliminación de los datos.

Autenticación Este servicio se refiere a la identificación tanto de las personas que establecen una comunicación, garantizando que cada una es quien dice ser; como del origen de la información que se maneja, garantizando la veracidad de la hora y fecha de origen, el contenido, tiempos de envío, entre otros.

No repudio Es el servicio que evita que el autor de la información o de alguna acción determinada, pueda negar su validez, ayudando así a prevenir situaciones de disputa.

2.1.1. Criptoanálisis y ataques

La criptografía forma parte de una ciencia más general llamada CRIPTOLOGÍA, la cual tiene otras ramas de estudio, como es el criptoanálisis que es la ciencia encargada de estudiar los posibles ataques a sistemas criptográficos, que son capaces de contrariar sus servicios ofrecidos. Entre los principales objetivos del criptoanálisis están interceptar la información que circula en un canal de comunicación, alterar dicha información y suplantar la identidad, rompiendo con los servicios de confidencialidad, integridad y autenticación respectivamente.

Los ataques que se realizan a sistemas criptográficos dependen de la cantidad de recursos o conocimientos con los que cuenta el adversario que realiza dicho ataque, dando como resultado la siguiente clasificación.

Ataque con sólo texto cifrado En este ataque el adversario solamente es capaz de obtener la información cifrada, y tratará de conocer su contenido en claro a partir de ella. Esta forma de atacar es la más básica, y todos los métodos criptográficos deben poder soportarla.

Ataque con texto en claro conocido Esta clase de ataques ocurren cuando el adversario puede obtener pares de información cifrada y su correspondiente información en claro, y por medio de su estudio, trata de descifrar otra información cifrada para la cual no conoce su contenido.

Ataque con texto en claro elegido Este ataque es muy parecido al anterior, con la diferencia de que en este el adversario es capaz de obtener los pares de información cifrada y en claro con el contenido que desee.

Ataque con texto en claro conocido adaptativo En este ataque el adversario es capaz de obtener los pares de información cifrada y en claro con el contenido que desee, además tiene amplio acceso o puede usar de forma repetitiva el mecanismo de cifrado.

Ataque con texto en claro elegido adaptativo En este caso el adversario puede elegir información cifrada y conocer su contenido, dado que tiene acceso a los mecanismos de descifrado.

2.1.2. Clasificación de la criptografía

La criptografía puede clasificarse de forma histórica en dos categorías, la criptografía clásica y la criptografía moderna. La criptografía clásica es aquella que se utilizó desde la antigüedad, teniéndose registro de su uso desde hace más 4000 años por los egipcios, hasta la mitad del siglo XX. En esta los métodos utilizados para cifrar eran variados, pero en su mayoría usaban la transposición y la sustitución, además de que la mayoría se mantenían en secreto. Mientras que la criptografía moderna es la que se inició después la publicación de la *Teoría de la información* por Claude Elwood Shannon[21], dado que esta sentó las bases matemáticas para la CRIPTOLOGÍA en general.

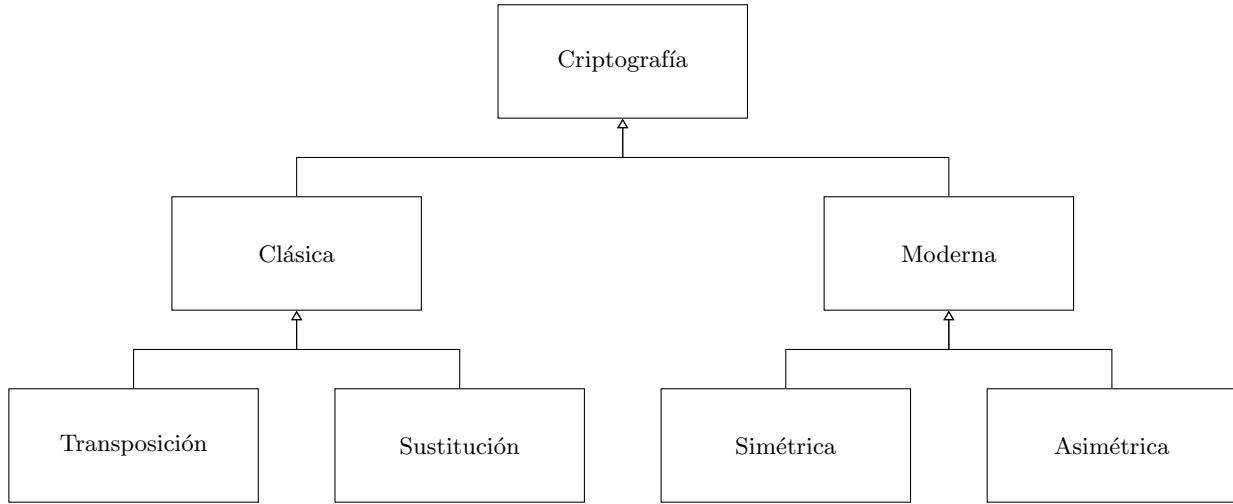


Figura 2.1: Clasificación de la criptografía.

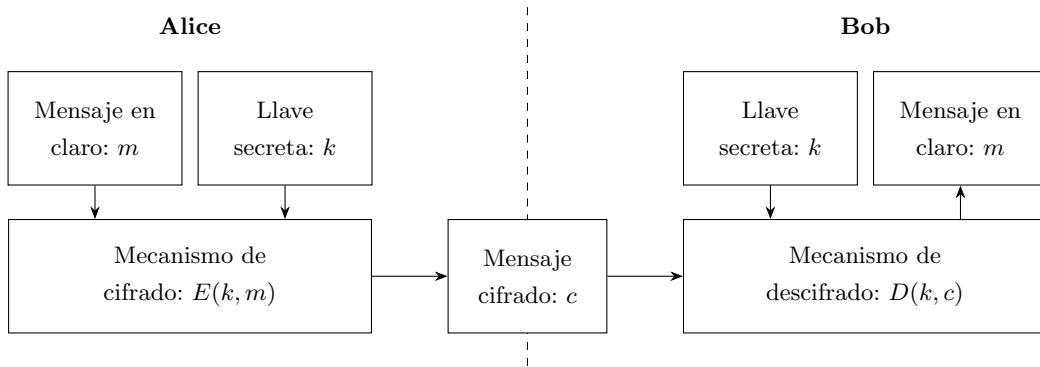


Figura 2.2: Canal de comunicación con criptografía simétrica.

Una manera de clasificar es de acuerdo a las técnicas y métodos empleados para cifrar la información, esta clasificación se puede observar en la figura 2.1.

Adentrándose en la clasificación de la criptografía clásica, se tienen los cifrados por transposición, los cuales se basan en técnicas de PERMUTACIÓN de forma que los caracteres de la información en claro se reordenen mediante algoritmos específicos, y los cifrados por sustitución, que utilizan técnicas de modificación de los caracteres por otros correspondientes a un alfabeto específico para el cifrado.

En cuanto a la criptografía moderna, esta tiene dos vertientes: la criptografía simétrica o de llave secreta, y la asimétrica o de llave pública. Hablando de la primer vertiente, se puede decir que es aquella que utiliza un modelo matemático para cifrar y descifrar un mensaje utilizando únicamente una llave que permanece secreta.

En la figura 2.2 se puede observar el proceso para establecer una comunicación segura por medio de

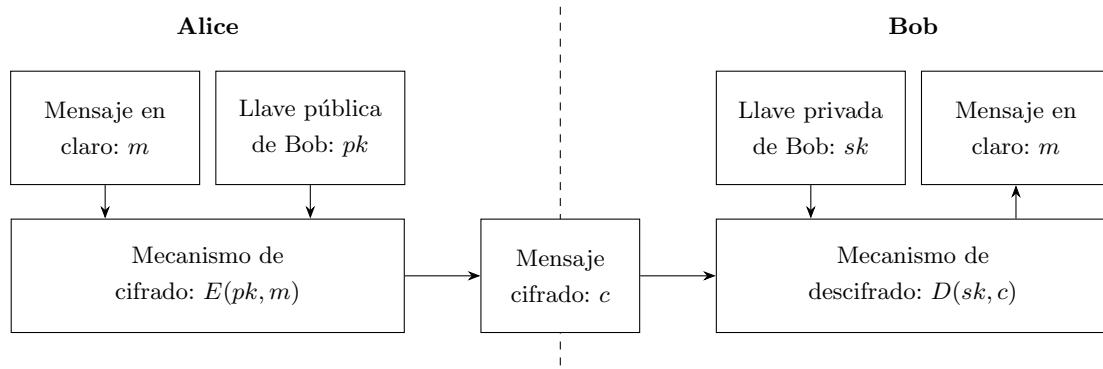


Figura 2.3: Canal de comunicación con criptografía asimétrica.

la criptografía simétrica. Primero, tanto Alice como Bob deben de establecer una llave única y compartida k , para que después, Alice, actuando como el emisor, cifre un mensaje m usando la llave k por medio del algoritmo de cifrado $E(k, m)$ para obtener el mensaje cifrado c y enviárselo a Bob. Posteriormente Bob, como receptor, se encarga de descifrar c con ayuda de la llave k por medio del algoritmo de descifrado $D(k, c)$ para obtener el mensaje original m .

Gran parte de los algoritmos de cifrado que caen en este tipo de criptografía están basados en las redes Feistel, que son un método de cifrado propuesto por el criptógrafo Horst Feistel, mismo que desarrolló el DATA ENCRYPTION STANDARD (DES) (sección 2.2.4) a principios de la década de los 70, que fue el cifrado usado por el gobierno estadounidense hasta 2002, año en que el ADVANCED ENCRYPTION STANDARD (AES) (sección 2.2.5) lo sustituyó.

Ahora, adentrándose en la criptografía asimétrica, se tiene que su idea principal es el uso de 2 llaves distintas para cada persona, una llave pública para cifrar, que esté disponible para cualquier otra persona, y una llave privada para descifrar, que se mantiene disponible solo para su propietario.

El proceso para establecer una comunicación segura por medio de este tipo de criptografía es el siguiente: primero, Alice nuevamente como el emisor, cifra un mensaje m con la llave pública de Bob pk , y usa el algoritmo de cifrado $E(pk, m)$ para obtener c y enviarlo. Después Bob como receptor, se encarga de descifrar c por medio del algoritmo de descifrado $D(sk, c)$ haciendo uso de su llave privada sk . Este proceso se refleja gráficamente en la figura 2.3.

Entre los usos que se le da a esta criptografía está el mantener la distribución de llaves privadas segura y establecer métodos que garanticen la autenticación y el no repudio; por ejemplo, en las firmas y certificados digitales.

2.2. Cifrados por bloques

La información presentada a continuación puede consultarse con más profundidad en las siguientes referencias [19], [20].

Los cifrados por bloque son esquemas de cifrado que, como bien lo explica su nombre, operan mediante bloques de datos. Normalmente los bloques tienen una longitud de 64 o de 128 bits, mientras que las llaves pueden ser de 56, 128, 192 o 256 bits.

En muchos sistemas criptográficos, los cifrados por bloque simétricos son elementos importantes, pues su versatilidad permite construir con ellos generadores de números pseudoaleatorios, cifrados de flujo MACs y funciones hash. Sirven también como componentes centrales en técnicas de autenticación de mensajes, mecanismos de integridad de datos, protocolos de autenticación de entidad y esquemas de firma electrónica que usan llaves simétricas.

Los cifrados por bloque están limitados en la práctica por varios factores, tales como el límite de memoria, la velocidad requerida o restricciones impuestas por el hardware o el software en el que se implementan. Normalmente, se debe escoger entre eficiencia y seguridad.

Idealmente, al cifrar por bloques, cada bit del bloque cifrado depende de todos los bits de la llave y del texto en claro; no debería existir una relación estadística evidente entre el texto en claro y el texto cifrado; el alterar tan solo un bit en el texto en claro o en la llave debería alterar cada uno de los bits del texto cifrado con una probabilidad de $\frac{1}{2}$; y alterar un bit del texto cifrado debería provocar resultados impredecibles al recuperar el texto en claro.

2.2.1. Definición

$$\begin{aligned} E : \{0,1\}^r \times \{0,1\}^n &\longrightarrow \{0,1\}^n \\ (k, m) &\longmapsto E(k, m) \end{aligned} \tag{2.1}$$

Utilizando una llave secreta k de longitud binaria r el algoritmo de cifrado E cifra bloques en claro m de una longitud binaria fija n y da como resultado bloques cifrados $c = E(k, m)$ cuya longitud también es n . n es el tamaño de bloque del cifrado. El espacio de llave está dado por $K = \{0,1\}^r$, para cada llave existe una función $D_k(c)$ que permite tomar un bloque cifrado c y regresarlo a su forma original m .

Generalmente, los cifrados por bloque procesan el texto claro en bloques relativamente grandes ($n \geq 64$), contrastando con los cifrados de flujo, que toman bit por bit. Cuando la longitud del mensaje en claro excede el tamaño de bloque, se utilizan los MODOS DE OPERACIÓN.

Los parámetros más importantes de los cifrados por bloque son los siguientes:

- Tamaño de bloque
- Tamaño de llave

2.2.2. Criterios para evaluar los cifrados por bloque

A continuación se listan algunos de los criterios que pueden ser tomados en cuenta para evaluar estos cifrados:

1. **Nivel de seguridad.** La confianza que se le tiene a un cifrado va creciendo con el tiempo, pues va siendo analizado y sometido a pruebas.
2. **Tamaño de llave.** La ENTROPÍA del espacio de la llave define un límite superior en la seguridad del cifrado al tomar en cuenta la búsqueda exhaustiva. Sin embargo, hay que tener cuidado con su tamaño, pues también aumentan los costos de generación, transmisión, almacenamiento, etcétera.
3. **Tamaño de bloque.** Impacta la seguridad, pues entre más grandes, mejor; sin embargo, tiene repercusiones en el costo de la implementación, además de que puede afectar el rendimiento del cifrado.
4. **Expansión de datos.** Es extremadamente deseable que los datos cifrados no aumenten su tamaño respecto a los datos en claro.
5. **Propagación de error.** Descifrar datos que contienen errores de bit puede llevar a recuperar incorrectamente el texto en claro, además de propagar errores en los bloques pendientes por descifrar. Normalmente, el tamaño de bloque afecta el error de propagación.

Entre los algoritmos de cifrado por bloque más usados se encuentran: DATA ENCRYPTION STANDARD (DES), ADVANCED ENCRYPTION STANDARD (AES), FAST DATA ENCIPHERMENT ALGORITHM (FEAL), INTERNATIONAL DATA ENCRYPTION ALGORITHM (IDEA), SECURE AND FAST ENCRYPTION ROUTINE (SAFER) y RC5. A continuación se presentan las redes Feistel, las cuales se utilizan para la construcción de otros cifrados por bloques, DES y AES.

2.2.3. Redes Feistel

Consiste en un CIFRADO ITERATIVO que mapea bloques de texto en claro de tamaño $2t$ bits (separados en bloques L_0, R_0 de tamaño t) a un texto cifrado R_r, L_r mediante un proceso de r RONDAS.

Cada subllave K_i se obtiene de la llave K . Normalmente el número de rondas r es mayor o igual a tres y par. Además, casi siempre intercambia el orden de los bloques de salida al revés en la última ronda: (R_r, L_r) en vez de (L_r, R_r) . El descifrado se realiza utilizando el mismo proceso de cifrado pero con las llaves en el orden inverso (comenzando con K_r hasta K_1).

```

1  inicio
2  para_todo  $i$  desde 1 hasta  $r$ :
3       $L_i = R_{i-1}$ 
4       $R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$ 
5  fin
6  fin

```

Pseudocódigo 2.1: Feistel, cifrado.

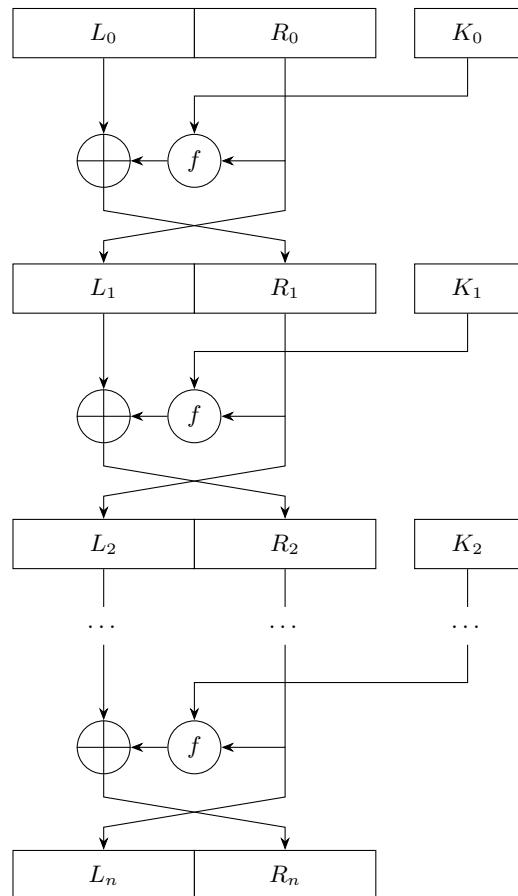


Figura 2.4: Diagrama genérico de una red Feistel.

Esta versión de las redes Feistel tiene como restricción que la longitud del bloque a procesar debe ser par. Existen dos generalizaciones del esquema original que permiten procesar bloques de cualquier longitud: las redes Feistel desbalanceadas y las redes Feistel alternantes (figura 2.5) [22].

Redes Feistel desbalanceadas

Este tipo de redes (presentadas en [23]) permite que los bloques izquierdos y derechos sean de distintas longitudes (m y n , respectivamente). En la figura 2.5A se muestra un esquema del proceso, el cual es bastante similar al de la figura 2.4 con la única diferencia de que la función f debe cambiar la longitud de su entrada: de n a m . Si $m \leq n$, la red es pesada del lado de la fuente, y la función actúa como contracción (la entrada es más grande que la salida). Por otra parte, si $m \geq n$, la red es pesada del lado del objetivo, y la función actúa como una expansión (la entrada es más pequeña que la salida). El caso especial en el que $m = n$ es en el que la red está balanceada y corresponde al presentado originalmente (figura 2.4); es por esto que las redes Feistel desbalanceadas son consideradas una generalización del esquema inicial.

Para los esquemas pesados del lado de la fuente, la seguridad aumenta proporcionalmente al grado de desbalanceo. Por el lado contrario, en los esquemas pesados del lado del objetivo, entre más balanceada la red, mejor.

Redes Feistel alternantes

Un inconveniente de las redes desbalanceadas es el costo extra de hacer las particiones de los bloques intermedios. Las redes Feistel alternantes (figura 2.5B, presentadas en [24] y [25]) eliminan este inconveniente utilizando dos tipos de funciones, una contractor y la otra de expansión, en RONDAS alternas.

Es importante notar que las redes alternantes son también una generalización del esquema original, en la cual la partición de los bloques es al centro, y se utiliza una sola función.

2.2.4. DATA ENCRYPTION STANDARD (DES)

Este es, probablemente, el cifrado simétrico por bloques más conocido; ya que en la década de los 70 estableció un precedente al ser el primer algoritmo a nivel comercial que publicó abiertamente sus especificaciones y detalles de implementación. Se encuentra definido en el estándar americano FEDERAL INFORMATION PROCESSING STANDARD (FIPS) 46-2.

DATA ENCRYPTION STANDARD (DES) es un cifrado Feistel que procesa bloques de $n = 64$ bits y produce bloques cifrados de la misma longitud. Aunque la llave es de 64 bits, 8 son de paridad, por lo que el tamaño *efectivo* de la llave es de 56 bits. Las 2^{56} llaves implementan, máximo, 2^{56} de las $2^{64}!$ posibles BIYECCIONES en bloques de 64 bits.

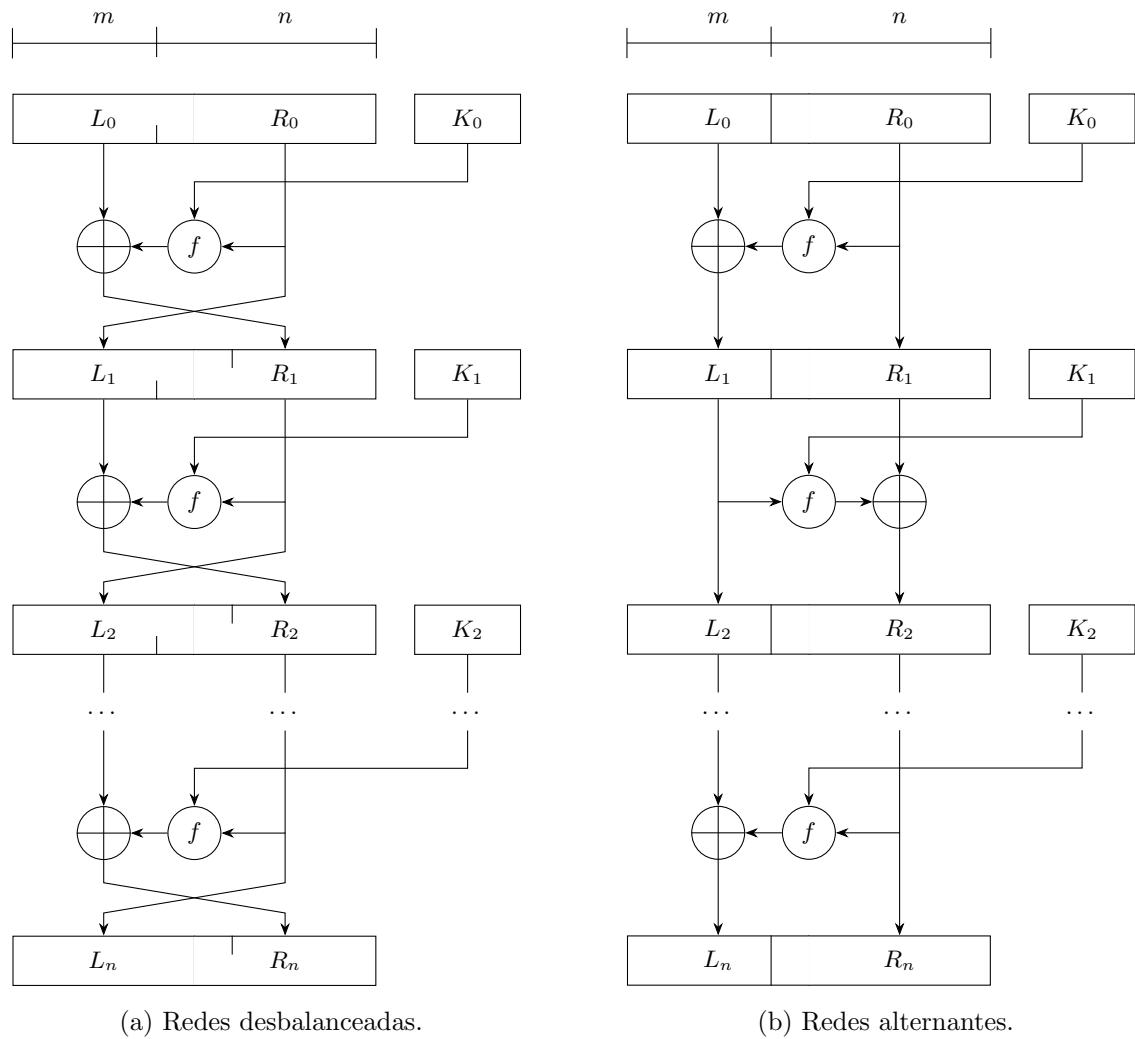


Figura 2.5: Generalizaciones de las redes Feistel.

```

1   entrada: 64 bits de texto en claro  $M = m_1 \dots m_{64}$  ;
2           llave de 64 bits  $K = k_1 \dots k_{64}$  .
3   salida: bloque de texto cifrado de 64 bits  $C = c_1 \dots c_{64}$  .
4   inicio
5       Calcular 16 subllaves  $K_i$  de 48 bits partiendo de  $K$  .
6       Obtener  $(L_0, R_0)$  de la tabla de permutaciones iniciales  $IP(m_1 m_2 \dots m_{64})$ 
7       para_todo  $i$  desde 1 hasta 16:
8            $L_{-i} = R_{-\{i-1\}}$ 
9           Obtener  $f(R_{i-1}, K_i)$  :
10          a) Expandir  $R_{i-1} = r_1 r_2 \dots r_{32}$  de 32 a 48 bits
11             usando  $E$ :  $T \leftarrow E(R_{i-1})$  .
12          b)  $T' \leftarrow T \oplus K_i$ . Donde  $T'$  es representado
13             como ocho cadenas de 6 bits cada una  $(B_1, \dots, B_8)$  .
14          c)  $T'' \leftarrow (S_1(B_1), S_2(B_2), \dots, S_8(B_8))$ 
15          d)  $T''' \leftarrow P(T'')$ 
16            $R_{-i} = L_{-\{i-1\}} \oplus f(R_{-\{i-1\}}, K_{-i})$ 
17   fin
18    $b_1 b_2 \dots b_{64} \leftarrow (R_{16}, L_{16})$  .
19    $C \leftarrow IP^{-1}(b_1 b_2 \dots b_{64})$ 
20   fin

```

Pseudocódigo 2.2: DES, cifrado.

Con la llave K se generan 16 subllaves K_i de 48 bits; una para cada RONDA. En cada RONDA se utilizan 8 *cajas-s* (mapeos de sustitución de 6 a 4 bits). La entrada de 64 bits es dividida por la mitad en L_0 y R_0 . Cada RONDA i va tomando las entradas L_{i-1} y R_{i-1} de la RONDA anterior y produce salidas de 32 bits L_i y R_i mientras $1 \leq i \leq 16$ de la siguiente manera:

$$\begin{aligned}
 L_i &= R_{i-1} \\
 R_i &= L_{i-1} \oplus f(R_{i-1}, K_i) \\
 \text{donde } f(R_{i-1}, K_i) &= P(S(E(R_{i-1}) \oplus K_i))
 \end{aligned} \tag{2.2}$$

E se encarga de expandir R_{i-1} de 32 bits a 48, P es una permutación de 32 bits y S son las cajas-s.

El descifrado DES consiste en el mismo algoritmo de cifrado, con la misma llave K , pero utilizando las subllaves en orden inverso: $K_{16}, K_{15}, \dots, K_1$.

Llaves débiles

Tomando en cuenta las siguientes definiciones:

- Llave débil: una llave K tal que $E_K(E_K(M)) = M$ para toda x ; en otras palabras, una llave débil permite que, al cifrar dos veces con la misma llave, se obtenga de nuevo el mensaje en claro.
- Llaves semidébiles: se tiene un par de llaves K_1, K_2 tal que $E_{K_1}(E_{K_2}(x)) = x$.

DES tiene cuatro llaves débiles y seis pares de llaves semidébiles. Las cuatro llaves débiles generan subllaves K_i iguales y, debido a que DES es un cifrado Feistel, el cifrado es autorreversible. O sea que al final se obtiene de nuevo el texto en claro, pues cifrar dos veces con la misma llave regresa la entrada original. Respecto a los pares semidébiles, el cifrado con una de las llaves del par es equivalente al descifrado con la otra (o viceversa).

2.2.5. ADVANCED ENCRYPTION STANDARD (AES)

Dado que el tamaño de bloque y la longitud de la llave de DATA ENCRYPTION STANDARD (DES) se volvieron muy pequeños para resistir los embates del progreso de la tecnología, el NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY (NIST) comenzó la búsqueda de un nuevo cifrado estándar en 1997; este cifrado debía tener un tamaño de bloque de, al menos, 128 bits y soportar tres tamaños de llave: 128, 192 y 256 bits.

Después de pasar por un proceso de selección, la propuesta Rijndael fue seleccionada. Se le hicieron algunas modificaciones, pues Rijndael soporta combinaciones de llaves y bloques de longitud 128, 169, 192, 224 y 256; mientras que ADVANCED ENCRYPTION STANDARD (AES) tiene fijo el tamaño de bloque y solo utiliza los tres tamaños de llave mencionados anteriormente. Dependiendo del tamaño de la llave, se tiene el número de RONDAS: 10 para las de 128 bits, 12 para las de 192 y 14 para las de 256.

El cifrado requiere de una matriz de 4×4 denominada matriz de estado.

Como todos los pasos realizados en las RONDAS son invertibles, el proceso de descifrado consiste en aplicar las funciones inversas a *SubBytes*, *ShiftRows*, *MixColumns* y *AddRoundKey* en el orden opuesto. Tanto el algoritmo como sus pasos están pensados con bytes. En el algoritmo Rijndael los bytes son considerados como elementos del campo finito \mathbb{F}_{2^8} con 2^8 elementos; \mathbb{F}_{2^8} es construido como una extensión del campo \mathbb{F}_2 con 2 elementos mediante el uso del polinomio irreducible $X^8 + X^4 + X^3 + X + 1$. Por lo tanto, las operaciones que se hagan a continuación de adición y el producto entre bytes significa sumarlos y multiplicarlos como elementos del campo \mathbb{F}_{2^8} .

```

1 entrada: 128 bits de texto en claro  $M$ ; llave de  $n$  bits  $K$ .
2 salida: bloque de texto cifrado de 64 bits  $C = c_1 \dots c_{64}$ .
3 inicio
4     Obtener las subllaves de 128 bits necesarias: una para cada ronda y una extra.
5     Iniciar matriz de estado con el bloque en claro.
6     Realizar  $AddRoundKey(\text{matriz\_estado}, k_0)$ 
7     para_todo  $i$  desde 1 hasta  $\text{num\_rondas} - 1$ :
8          $\text{SubBytes}(\text{matriz\_estado})$ 
9          $\text{ShiftRows}(\text{matriz\_estado})$ 
10         $\text{MixColumns}(\text{matriz\_estado})$ 
11         $\text{AddRoundKey}(\text{matriz\_estado}, k_i)$ 
12     fin
13      $\text{SubBytes}(\text{matriz\_estado})$ 
14      $\text{ShiftRows}(\text{matriz\_estado})$ 
15      $\text{MixColumns}(\text{matriz\_estado})$ 
16      $\text{AddRoundKey}(\text{matriz\_estado}, k_{\text{num\_rondas}})$ 
17     regresa  $\text{matriz\_estado}$ 
18 fin

```

Pseudocódigo 2.3: AES, cifrado.

SubBytes

Esta es la única transformación no lineal de Rijndael. Sustituye los bytes de la matriz de estado byte a byte al aplicar la función S_{RD} a cada elemento de la matriz. La función S_{RD} es también conocida como Caja-S y no depende de la llave. La misma caja es utilizada para los bytes en todas las posiciones.

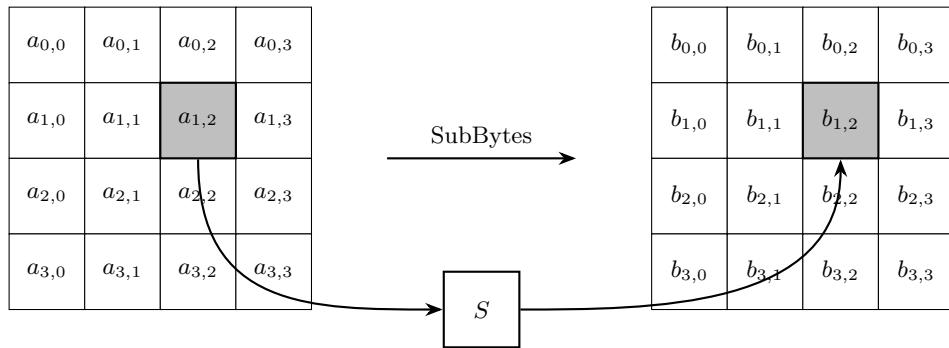
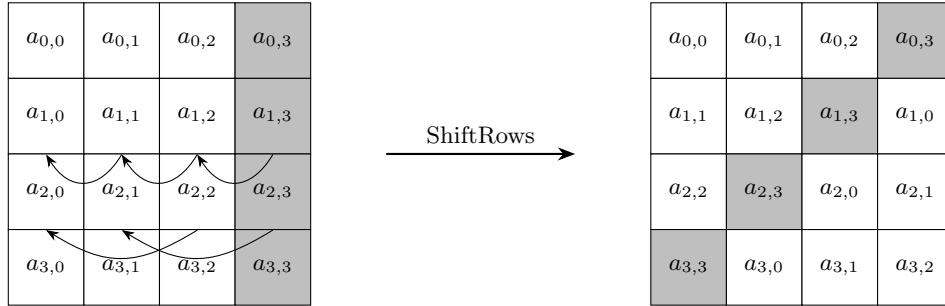
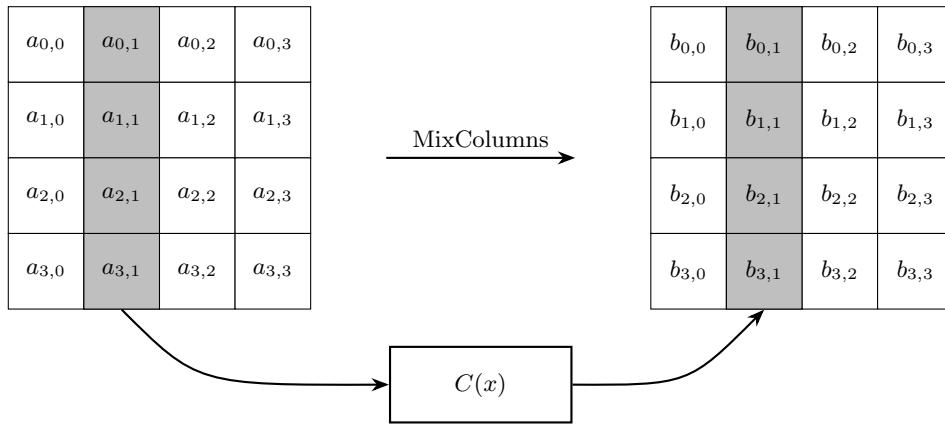


Figura 2.6: Diagrama de la operación SubBytes .


 Figura 2.7: Diagrama de la operación *ShiftRows*.

 Figura 2.8: Diagrama de la operación *MixColumns*.

ShiftRows

Esta transformación hace un corrimiento cíclico hacia la izquierda de las filas de la matriz de estado. Los desplazamientos son distintos para cada fila y dependen de la longitud del bloque (N_b).

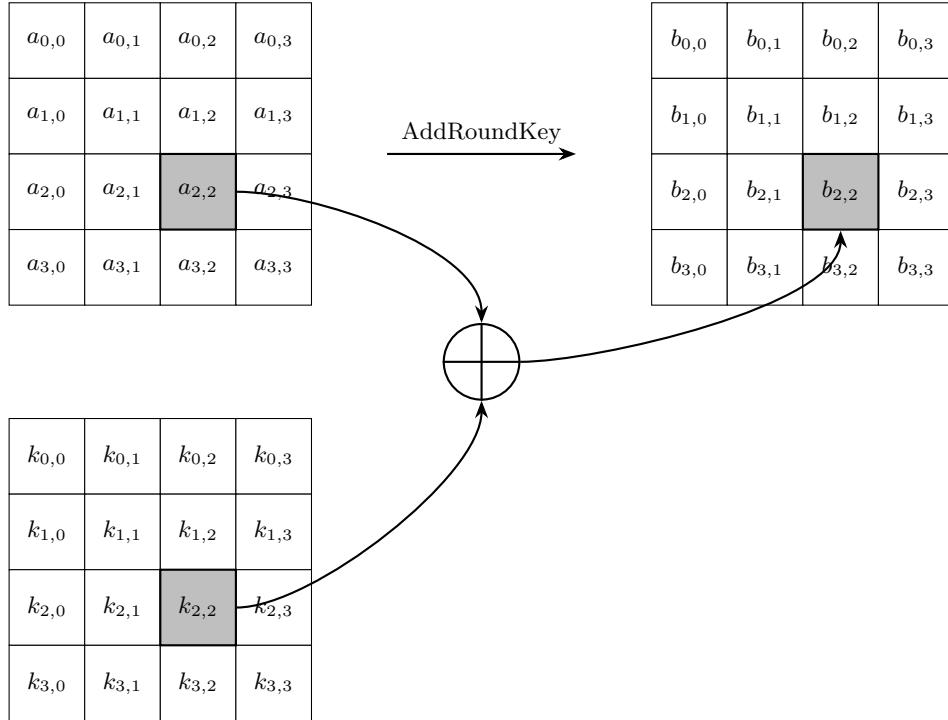
MixColumns

Esta transformación opera en cada columna de la matriz de estado independientemente. Se considera una columna $a = (a_0, a_1, a_2, a_3)$ como el polinomio $a(X) = a_3X^3 + a_2X^2 + a_1X + a_0$. Entonces este paso transforma una columna a al multiplicarla con el siguiente polinomio fijo:

$$c(X) = 03X^3 + 01X^2 + 01X + 02 \quad (2.3)$$

y se toma el residuo del producto módulo $X^4 + 1$:

$$a(X) \mapsto a(X) \cdot c(X) \pmod{X^4 + 1} \quad (2.4)$$


 Figura 2.9: Diagrama de la operación *AddRoundKey*.

AddRoundKey

Esta es la única operación que depende de la llave secreta k . Añade una llave de ronda para intervenir en el resultado de la matriz de estado. Las llaves de ronda son derivadas de la llave secreta k al aplicar el algoritmo de generación de llaves. Las llaves de ronda tienen la misma longitud que los bloques. Esta operación es simplemente una operación *XOR* bit a bit de la matriz de estado con la llave de ronda en turno. Para obtener el nuevo valor de la matriz de estado se realiza lo siguiente:

$$(matriz_estado, k_i) \mapsto matriz_estado \oplus k_i \quad (2.5)$$

Como se tiene una matriz_estado, la llave de ronda (k_i) también es representada como una matriz de bytes con 4 columnas y N_b columnas. Cada una de las N_b palabras de la llave de ronda corresponde a una columna. Entonces se realiza la operación *XOR* bit a bit sobre las entradas correspondientes de la matriz de estado y la matriz de la llave de ronda.

Esta operación, claro está, es invertible: basta con aplicar la misma operación con la misma llave para revertir el efecto.

```
1 entrada: llave  $k$ ; bloques de mensaje  $Bm_1, Bm_2 \dots Bm_n$ .
2 salida: bloques de mensaje cifrado  $Bc_1, Bc_2 \dots Bc_n$ .
3 inicio
4   para_todo  $Bm$ 
5      $Bc_i \leftarrow E_k(Bm_i)$ 
6   fin
7   regresar  $Bc$ 
8 fin
```

Pseudocódigo 2.4: MODO DE OPERACIÓN ECB, cifrado.

2.2.6. Modos de operación

La información que aquí se presenta se puede consultar a mayor detalle en [26] y [19].

Por sí solos, los cifrados por bloques solamente permiten el cifrado y descifrado de bloques de información de tamaño fijo; donde, en la mayoría de los casos, los bloques son de menos de 256 bits, lo cual es equivalente a alrededor de 8 caracteres. Es fácil darse cuenta de que esta restricción no es ningún tema menor: en la gran mayoría de las aplicaciones, la longitud de lo que se quiere ocultar es arbitraria.

Los MODOS DE OPERACIÓN permiten extender la funcionalidad de los cifrados por bloques para poder aplicarlos a información de tamaño irrestrictivo: reciben el texto original (de tamaño arbitrario) y lo cifran, ocupando en el proceso un cifrado por bloques.

Un primer enfoque (y quizás el más intuitivo) es partir el mensaje original en bloques del tamaño requerido y después aplicar el algoritmo a cada bloque por separado; en caso de que la longitud del mensaje no sea múltiplo del tamaño de bloque, se puede agregar información extra al último bloque para completar el tamaño requerido. Este es, de hecho, uno de los modos de operación que existen, el ELECTRONIC CODEBOOK (ECB); su uso no es recomendado, pues es muy inseguro cuando el mensaje original es simétrico a nivel de bloque.

A pesar de que existen más modos de operación, como CIPHER FEEDBACK (CFB) u OUTPUT FEEDBACK (OFB), en este trabajo solo se describirá el funcionamiento de ECB, CIPHER-BLOCK CHAINING (CBC) y COUNTER MODE (CTR), dado que son los modos de operación necesarios para el proceso de tokenización.

ELECTRONIC CODEBOOK (ECB)

La figura 2.10 muestra un diagrama esquemático de este MODO DE OPERACIÓN. El algoritmo recibe a la entrada una llave y un mensaje de longitud arbitraria: la llave se pasa sin ninguna modificación a cada función del cifrado por bloques; el mensaje se debe de partir en bloques ($M = Bm_1 || Bm_2 || \dots || Bm_n$).

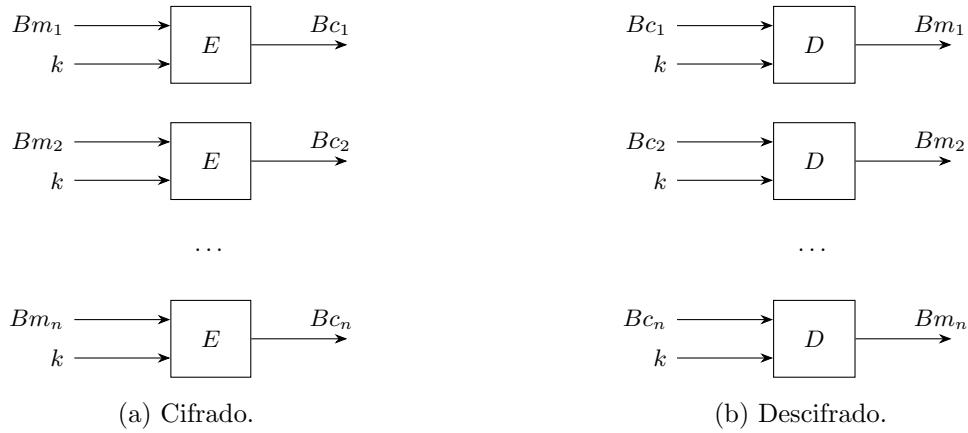


Figura 2.10: MODO DE OPERACIÓN ECB.

```

1 entrada: llave  $k$ ; bloques de mensaje cifrado  $Bc_1, Bc_2 \dots Bc_n$ .
2 salida: bloques de mensaje original  $B_1, B_2 \dots B_n$ .
3 inicio
4   para_todo  $Bc$ 
5      $Bm_i \leftarrow D_k(Bc_i)$ 
6   fin
7   regresar  $Bm$ 
8 fin

```

Pseudocódigo 2.5: MODO DE OPERACIÓN ECB, descifrado.

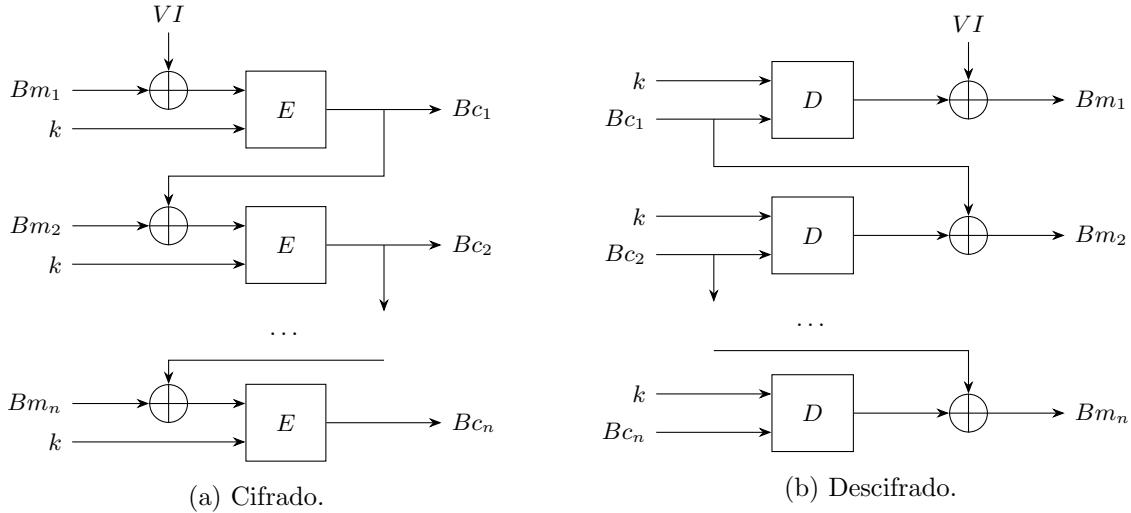


Figura 2.11: MODO DE OPERACIÓN CBC.

```

1 entrada: llave  $k$ ; vector de inicialización  $VI$ ;
2                 bloques de mensaje  $Bm_1, Bm_2 \dots Bm_n$ .
3 salida: bloques de mensaje cifrado  $Bc_1, Bc_2 \dots Bc_n$ .
4 inicio
5    $Bc_0 \leftarrow VI$                                 // El vector de inicialización
6   para_todo  $Bm_i$                             // entra al primer bloque.
7      $Bc_i \leftarrow E(k, Bm_i \oplus Bc_{i-1})$ 
8   fin
9   regresar  $Bc$ 
10 fin

```

Pseudocódigo 2.6: MODO DE OPERACIÓN CBC, cifrado.

CIPHER-BLOCK CHAINING (CBC)

En CBC la salida del bloque uno se introduce (junto con el siguiente bloque del mensaje) en el bloque dos, y así en sucesivo. Para poder replicar este comportamiento en todos los bloques, este MODO DE OPERACIÓN necesita un argumento extra a la entrada: un VECTOR DE INICIALIZACIÓN. De esta manera la salida del bloque i depende de todos los bloques anteriores; esto incrementa la seguridad con respecto a ECB.

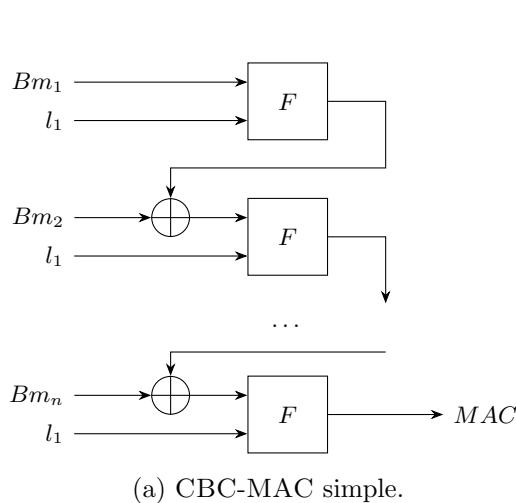
En la figura 2.11 se muestran los diagramas esquemáticos para cifrar y descifrar; en los pseudocódigos 2.6 y 2.7 se muestran unos de los posibles algoritmos a seguir. Es importante notar que mientras que el proceso de cifrado debe ser forzosamente secuencial (por las dependencias entre salidas), el proceso de descifrado puede ser ejecutado en paralelo.

```

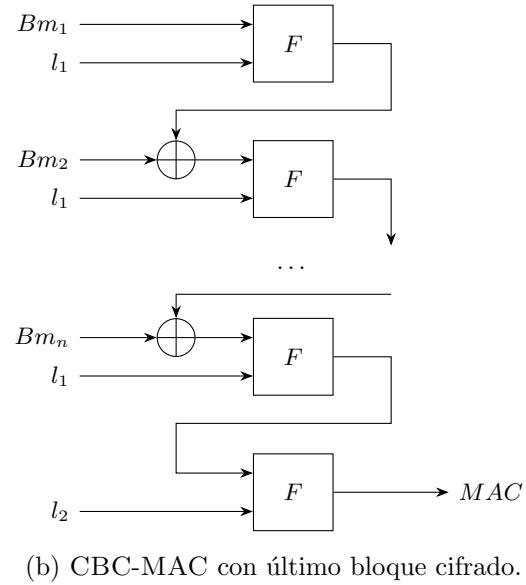
1 entrada: llave  $k$ ; vector de inicialización  $VI$ ;
2           bloques de mensaje cifrado  $Bc_1, Bc_2 \dots Bc_n$ .
3 salida: bloques de mensaje original  $Bm_1, Bm_2 \dots Bm_n$ .
4 inicio
5    $Bc_0 \leftarrow VI$ 
6   para_todo  $Bc$ 
7      $Bm_i \leftarrow D_k(Bc_i) \oplus Bc_{i-1}$ 
8   fin
9   regresar  $Bm$ 
10 fin

```

Pseudocódigo 2.7: MODO DE OPERACIÓN CBC, descifrado.



(a) CBC-MAC simple.



(b) CBC-MAC con último bloque cifrado.

Figura 2.12: CBC-MAC.

CBC-MAC Este algoritmo está basado en el modo de operación CBC y una función F que puede ser, por ejemplo, un cifrado por bloques. Se encarga de cifrar con una llave l_1 todo el mensaje m , pero lo único que se toma en cuenta es el último bloque, que es tomado como el código de autenticación (véase figura 2.12A). En algunos casos, se cifra de nuevo, con la misma función F , pero utilizando una llave l_2 distinta (véase figura 2.12B).

CBC-MAC es considerado seguro cuando el mensaje m tiene una longitud múltiplo del tamaño de bloque del cifrado por bloques. Como se le han encontrado varias vulnerabilidades, algoritmos basados en CBC-MAC han sido propuestos, tales como el *eXtended CBC* o el ONE-KEY MAC (OMAC).

```

1 entrada: llave  $k$ ; vector de inicialización  $VI$ ;
2           bloques de mensaje (cifrado o descifrado)  $Bm_1, Bm_2 \dots Bm_n$ .
3 salida: bloques de mensaje (cifrado o descifrado)  $Bc_1, Bc_2 \dots Bc_n$ .
4 inicio
5   para_todo  $Bm$ 
6      $Bc_i \leftarrow E_k((VI + i) \bmod 2^n) \oplus Bm_i$ 
7   fin
8   regresar  $Bc$ 
9 fin

```

Pseudocódigo 2.8: MODO DE OPERACIÓN CTR(cifrado y descifrado).

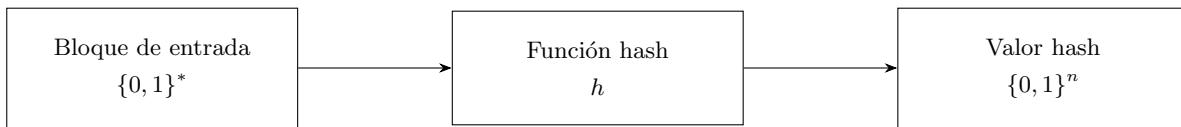


Figura 2.13: Diagrama del funcionamiento de una función hash.

COUNTER MODE (CTR)

Este toma a la entrada un VECTOR DE INICIALIZACIÓN y en cada iteración lo incrementa y lo cifra. El resultado se obtiene combinando el VI cifrado con el bloque de texto cifrado (mediante una operación xor). El proceso de cifrado y descifrado se detallan en el pseudocódigo 2.8.

En términos de eficiencia, el CTR es mejor que CBC, ya que sus operaciones (ambas) se pueden hacer en paralelo. La implementación es prácticamente la misma para el cifrado y descifrado (solamente se ocupa el cifrado del algoritmo por bloques subyacente).

2.3. Funciones hash

La información presentada a continuación puede consultarse con más profundidad en las siguientes referencias [19], [20], [27], [28].

Se refiere al conjunto de funciones computacionalmente eficientes que mapean cadenas binarias de una longitud arbitraria a cadenas binarias de una longitud fija, llamadas valores hash.

Matemáticamente, una función hash es una función

$$\begin{aligned}
 h : \{0,1\}^* &\longrightarrow \{0,1\}^n \\
 m &\longmapsto h(m)
 \end{aligned} \tag{2.6}$$

La longitud de n suele ser entre 128 y 512 bits. Las funciones hash h tienen las siguientes propiedades:

1. Compresión: h mapea una entrada x (cuya longitud finita es arbitraria) a una salida $h(x)$ de longitud fija n .
2. Facilidad de cómputo: dada x y h , $h(x)$ es calculada ya sea sin necesitar mucho espacio, tiempo de cómputo, o requiere pocas operaciones, etcétera.

De manera general, las funciones hash se pueden dividir en dos categorías: las que no utilizan llave y su único parámetro es la entrada x , y las que necesitan una llave secreta k y la entrada x .

Sea una función hash sin llave h con entradas x , x' y salidas y y y' , respectivamente. A continuación se listan algunas de las propiedades que puede tener:

1. Resistencia de PREIMAGEN: no es computacionalmente factible para una salida específica y encontrar una entrada x' que dé como resultado el mismo valor hash $h(x') = y$ si no se conoce x . Esta propiedad también es llamada *de un sentido*.
2. Resistencia de segunda PREIMAGEN: no es computacionalmente factible encontrar una segunda entrada x' que tenga la misma salida que una entrada específica x : $x \neq x'$ tal que $h(x) = h(x')$. Esta propiedad también es conocida como *débil resistencia a colisiones*.
3. Resistencia a las colisiones: no es computacionalmente factible encontrar dos entradas distintas x , x' que lleven al mismo valor hash, o sea, $h(x) = h(x')$. A diferencia de la anterior, la selección de ambas entradas no está restringida. Esta propiedad también es conocida como *gran resistencia a colisiones*.

Una función hash h que cumple con las propiedades de resistencia de PREIMAGEN y resistencia de segunda PREIMAGEN es conocida como una función hash de un solo sentido o ONE-WAY HASH FUNCTION (OWHF). Las que cumplen con la resistencia de segunda PREIMAGEN y resistencia a las colisiones son conocidas como funciones hash resistentes a colisiones o COLLISION-RESISTANT HASH FUNCTION (CRHF). Aunque casi siempre las funciones CRHF cumplen con la resistencia de PREIMAGEN, no es obligatorio que lo hagan.

Algunos ejemplos de las funciones OWHF son el SECURE HASH ALGORITHM (SHA)-1 y el MESSAGE DIGEST-5 (MD5). En los esquemas de firma electrónica, se obtiene el valor hash del mensaje ($h(m)$) y se pone en el lugar de la firma. Los valores hash también son utilizados para revisar la integridad de las llaves públicas y, al utilizarse con una llave secreta, las funciones criptográficas hash se convierten en códigos de autenticación de mensaje (MESSAGE AUTHENTICATION CODE (MAC), por sus siglas en inglés), una de las herramientas más utilizadas en protocolos como SECURE SOCKETS LAYER (SSL) e IPSec para revisar la integridad de un mensaje y autenticar al remitente.

Una de las aplicaciones más conocidas de las funciones hash es la de cifrar las contraseñas: en un sistema, en vez de almacenar la contraseña *clave*, se guarda su valor hash $h(\text{clave})$. Así, cuando un usuario

ingresa su contraseña, el sistema calcula su valor hash y lo compara con el que se tiene guardado. Realizar esto ayuda a evitar que las contraseñas sean conocidas para los usuarios con privilegios, como pueden ser los administradores.

2.3.1. SECURE HASH ALGORITHM (SHA)

El algoritmo SECURE HASH ALGORITHM (SHA) fue publicado por NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY (NIST) y NATIONAL SECURITY AGENCY (NSA) en 1993; este algoritmo produce bloques de 160 bits y fue desarrollado para reemplazar al MESSAGE DIGEST-4 (MD4); sin embargo, poco después de haber sido publicado tuvo que ser quitado por problemas de seguridad. Actualmente, SHA es conocido como SHA-0.

En 1995, SHA-0 fue reemplazado por SHA-1; tiene una salida de la misma longitud que su predecesor y es una de las funciones hash más populares. Hay que destacar que la seguridad que brinda esta función es limitada, pues tiene el mismo nivel que un cifrado por bloques de 80 bits.

En 2002 NIST publicó tres funciones hash más: SHA-256, SHA-384 y SHA-512; esta familia de funciones hash es conocida como SHA-2 y fue desarrollada para cubrir la necesidad de una llave más grande para poder empatar su tamaño con ADVANCED ENCRYPTION STANDARD (AES). Dos años más tarde, una nueva función hash fue agregada a la familia SHA-2: SHA-224.

Finalmente, en 2008, NIST inició un concurso para buscar al SHA-3 y en 2012 anunció al ganador: Keccak, una función hash desarrollada por Guido Bertoni, Joan Daemen, Michael Peeters y Gilles Van Assche. Esta función tiene una construcción completamente distinta a las familias anteriores.

2.4. Cifrados que preservan el formato

El objetivo de los cifrados que preservan el formato (en inglés, FORMAT PRESERVING ENCRYPTION (FPE)) está bastante bien definido: lograr que los mensajes cifrados *se vean* igual que los mensajes en claro. Obviamente para que esto tenga sentido hay que definir qué es lo que hace que un mensaje se parezca a otro; los cifrados tradicionales, como ADVANCED ENCRYPTION STANDARD (AES), reciben cadenas binarias y entregan cadenas binarias, por lo que, en cierto sentido, son ya un tipo de cifrados que preservan el formato. En una clase más general de algoritmos FPE el formato debe ser una parámetro: cadenas binarias, caracteres AMERICAN STANDARD CODE FOR INFORMATION INTERCHANGE (ASCII) imprimibles, dígitos decimales, dígitos hexadecimales, etcétera.

Desde un inicio, los cifrados que preservan el formato se perfilaron como una posible solución para el problema de la tokenización: usando un alfabeto de dígitos decimales se logra que los *tokens* y los PERSONAL ACCOUNT NUMBER (PAN) tengan el mismo aspecto. De las posibles soluciones presentadas en la sección 3.2, FPE es la que presenta un esquema más tradicional dentro de la criptografía, ya que el

proceso de cifrado y descifrado es el mismo que un cifrado simétrico.

La utilidad de los cifrados que preservan el formato se centra principalmente en *agregar* seguridad a sistemas y protocolos que ya se encuentran en un entorno de producción. Estos son algunos ejemplos de dominios comunes en FPE:

- Números de tarjetas de crédito.

5827 5423 6584 2154 → 6512 8417 6398 7423

- Números de teléfono.

55 55 54 75 65 → 55 55 12 36 98

- CURP.

GHUJ887565HGBTOK01 → QRGH874528JUHY01

2.4.1. Clasificación de los cifrados que preservan el formato

En [22] Phillip Rogaway propone una clasificación para los cifrados que preservan el formato que se basa en el tamaño del espacio de mensajes ($N = |X|$):

Espacios minúsculos El espacio es tan pequeño que es aceptable gastar $O(N)$ en un preproceso de cifrado. Esto es, cifrar todos los posibles mensajes de una sola vez, para que las subsiguientes solicitudes de cifrado y descifrado consistan en simples consultas a una base de datos.

El tamaño de N depende del contexto en el que se vaya a utilizar el algoritmo; para el contexto del problema de la tokenización ($N \approx 10^{16}$) no resulta viable utilizar esta técnica.

Algunos ejemplos de cómo hacer el preproceso de cifrado son el *Knuth shuffle* (también conocido como *Fisher-Yates shuffle*, pseudocódigo 2.9) o un CIFRADO CON PREFIJO.

Espacios pequeños En esta clasificación se colocan a los espacios de mensaje cuyo tamaño no es más grande que 2^w en donde w es el tamaño de bloque del cifrado subyacente. Para AES, en donde $w = 128$, $N = 2^{128} \approx 10^{38}$.

En este esquema, el mensaje se ve como una cadena de n elementos pertenecientes a un alfabeto de cardinalidad m (esto es, $N = m^n$). Por ejemplo, para números de tarjetas de crédito, $n \approx 16$ y $m = 10$, por lo que $N = 10^{16}$ (diez mil trillones); lo cual es aproximadamente $2.93 \times 10^{-21}\%$ de 2^{128} .

Los algoritmos que preservan el formato expuestos en la sección 3.2 pertenecen a esta categoría.

Espacios grandes El espacio es más grande que 2^w . Para estos casos, el mensaje se ve como una cadena binaria. Las técnicas utilizadas incluyen cualquier cifrado cuya salida sea de la misma longitud que la entrada (p. ej. los TWEAKABLE ENCRYPTION SCHEME (TES): CBC-MASK-CBC (CMC), ECB-MASK-ECB (EME), HCH, etcétera).

```
1 entrada: lista  $l[0, \dots, n - 1]$ 
2 salida: misma lista barajeada
3 inicio
4   para_todo  $i$  desde  $n - 1$  hasta 0:
5      $j \leftarrow \text{rand}(0, i)$ 
6      $\text{swap}(l[j], l[i])$ 
7   fin
8 fin
```

Pseudocódigo 2.9: *Knuth shuffle*, [30].

Como se puede observar de los ejemplos dados, el problema de la tokenización de números de tarjetas de crédito es un problema de espacios pequeños.

En marzo de 2016 el NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY (NIST) publicó un estándar referente a los cifrados que preservan el formato[29]. En él se definen dos posibles métodos: FF1 (lo que en este documento es FORMAT-PRESERVING FEISTEL-BASED ENCRYPTION (FFX), sección 3.2.1) y FF3 (BRIER-PEYRIN-STERN (BPS), sección 3.2.2).

2.5. Composición del número de una tarjeta

La información presentada a continuación puede consultarse con más detalle en los siguientes documentos [31]-[33].

Como se puede observar en la figura 2.14, un número de tarjeta PERSONAL ACCOUNT NUMBER (PAN), se compone por tres partes: el número identificador del emisor (ISSUER IDENTIFICATION NUMBER (INN)), el número de cuenta y un dígito verificador; la longitud del PAN puede variar e ir desde los 12 hasta los 19 dígitos. A continuación se explica con más detalle cada uno de sus componentes.

2.5.1. Identificador del emisor

El ISSUER IDENTIFICATION NUMBER (INN) comprende los primeros 6 dígitos del número de tarjeta. El primer dígito es conocido como MAJOR INDUSTRY IDENTIFIER (MII) y se encarga de identificar la industria a la que pertenece el emisor; en la tabla 2.1, se puede observar la relación entre el dígito y el giro. El identificador del emisor provee, entre otros, los siguientes datos:

1. Emisor de la tarjeta
2. Tipo de la tarjeta (ej. crédito o débito)
3. Nivel de la tarjeta (ej. clásica, Gold, Black)

Dígito	Industria
1, 2	Aerolíneas
3	Viajes y entretenimiento (ej. American Express)
4, 5	Bancos e industria financiera (ej. Visa, Mastercard)
6	Comercio (ej. Discover)
7	Industria petrolera
8	Telecomunicaciones
9	Asignación nacional

Tabla 2.1: Identificador de industria (MII).

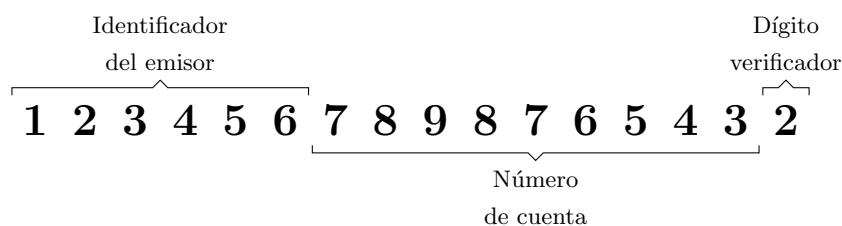


Figura 2.14: Componentes de un número de tarjeta.

2.5.2. Número de cuenta

Todos los dígitos posteriores al ISSUER IDENTIFICATION NUMBER (INN) y anteriores al último dígito, comprenden el número de cuenta. La longitud de este puede variar, pero, máximo comprende 12 dígitos, por lo que cada emisor tiene 10^{12} posibles números de cuenta.

2.5.3. Dígito verificador

Finalmente, se tiene el dígito verificador; este toma en cuenta todos los dígitos anteriores y se calcula mediante el algoritmo de Luhn, que se describe en el pseudocódigo 2.10.

2.6. Estándares del NIST y PCI-SSC

Primitivas criptográficas (Apéndice A)

Generación de bits pseudoaleatorios (Apéndice B)

800-90A Revision 1 *Recommendation for Random Number Generation Using Deterministic Random Bit Generators.* Disponible en [34].

Pruebas estadísticas para generadores pseudoaleatorios (Apéndice C)

800-22A Revision 1 *A Statistical Test Suite for Random and Pseudorandom Number Generators*

```

1   entrada: número de tarjeta , compuesto desde 12 hasta 19 dígitos:
2            $\{x_n x_{n-1} x_{n-2} \dots x_3 x_2 x_1\}$ 
3   salida: dígito verificador  $x_1$ .
4   inicio
5       Sean los conjuntos  $x_{par}$  y  $x_{impar}$ :
6       si  $n$  es par:
7            $x_{par} = \{x_2, x_4, x_6, \dots, x_n\}$ 
8            $x_{impar} = \{x_3, x_5, x_7, \dots, x_{n-1}\}$ .
9       sino :
10           $x_{par} = \{x_2, x_4, x_6, \dots, x_{n-1}\}$ 
11           $x_{impar} = \{x_3, x_5, x_7, \dots, x_n\}$ .
12   Obtener el doble de cada uno de los elementos del conjunto  $x_{par}$ :
13        $x_{pardoble} = \{2 \times x_2, 2 \times x_4, 2 \times x_6, \dots\}$ .
14        $\forall x_i \in x_{pardoble} > 9$ :
15            $x_i = (x_i \text{ mód } 10) + 1$ 
16   Sumar los elementos de los conjuntos  $x_{pardoble}$  y  $x_{impar}$ .
17    $x_1 = (S \times 9) \text{ mód } 10$ 
18   Regresar  $x_1$ 
19   fin

```

Pseudocódigo 2.10: Algoritmo de Luhn.

for Cryptographic Applications. Disponible en [35].

Generación de llaves (Apéndice D)

800-108 *Recommendation for Key Derivation Functions Using Pseudorandom Functions.* Disponible en [36].

800-133 *Recommendation for Cryptographic Key Generation.* Disponible en [37].

Administración de llaves (Apéndice E)

800-57 *Recommendation for Key Management.* Disponible en [38].

800-130 *A Framework for Designing Cryptographic Key Management Systems.* Disponible en [39].

Parte I

Generación de tokens

« “Begin at the beginning,” the King said,
very gravely, “and go on till you came to
the end: then stop.”»

LEWIS CARROLL.

Capítulo 3

Algoritmos tokenizadores

«Anyone who attempts to generate random numbers by arithmetical methods is, of course, living in a state of sin.»

Various techniques used in connection
with random digits, JOHN VON
NEUNMANN.

Este capítulo se clasifican los algoritmos que serán implementados y, para cada uno, se agrega una subsección donde se explica su notación, composición y funcionamiento.

3.1. Clasificación del PCI SSC

En [40], el PAYMENT CARD INDUSTRY (PCI) SECURITY STANDARD COUNCIL (SSC) divide a los TOKENS en reversibles e irreversibles. A su vez, los reversibles se dividen en criptográficos y en no criptográficos; mientras que los irreversibles se dividen en autenticables y no autenticables (figura 3.1).

Los TOKENS irreversibles no pueden, bajo ninguna circunstancia, ser reconvertidos al PERSONAL ACCOUNT NUMBER (PAN) original. Esta restricción aplica tanto para cualquier entidad en el entorno del negocio (comerciante, proveedor de TOKENS, banco) como para cualquier posible atacante. Dados un PAN y un TOKEN, los identificables permiten validar cuando el primero fue utilizado para la creación del segundo, mientras que los no identificables, no.

La clasificación del PCI SSC con respecto a los reversibles resulta un poco confusa: establece que los criptográficos son generados utilizando CRIPTOGRAFÍA FUERTE, el PAN nunca se almacena, solamente se guarda una llave; los no criptográficos guardan la relación entre TOKENS y PAN en una base de datos. El problema está en que no se menciona *cómo* generar los no criptográficos. A pesar del nombre, los métodos más comunes para esta categoría ocupan PRIMITIVAS CRIPTOGRÁFICAS (p. ej. generadores pseudoaleatorios); además de que, en una implementación real, para poder cumplir con el PCI DATA SECURITY STANDARD (DSS), la propia base de datos debe de estar cifrada [41].

3.2. Algoritmos tokenizadores

En esta sección se documentan distintos algoritmos para generar TOKENS; cada apartado incluye una descripción del algoritmo y su pseudocódigo. Antes de la documentación, se presentan dos clasificaciones de los algoritmos tokenizadores: la primera es la propuesta por el PAYMENT CARD INDUSTRY (PCI) SECURITY STANDARD COUNCIL (SSC) y mencionada en la sección 3.1 (figura 3.1); la segunda clasificación es propuesta por los autores de este documento, tomando en cuenta nociones criptográficas más estrictas (figura 3.2).

Clasificación del PCI SSC

- TOKENS reversibles:
 - TOKENS criptográficos:
 - FORMAT-PRESERVING FEISTEL-BASED ENCRYPTION (FFX) (véase sección 3.2.1).
 - BRIER-PEYRIN-STERN (BPS) (véase sección 3.2.2).

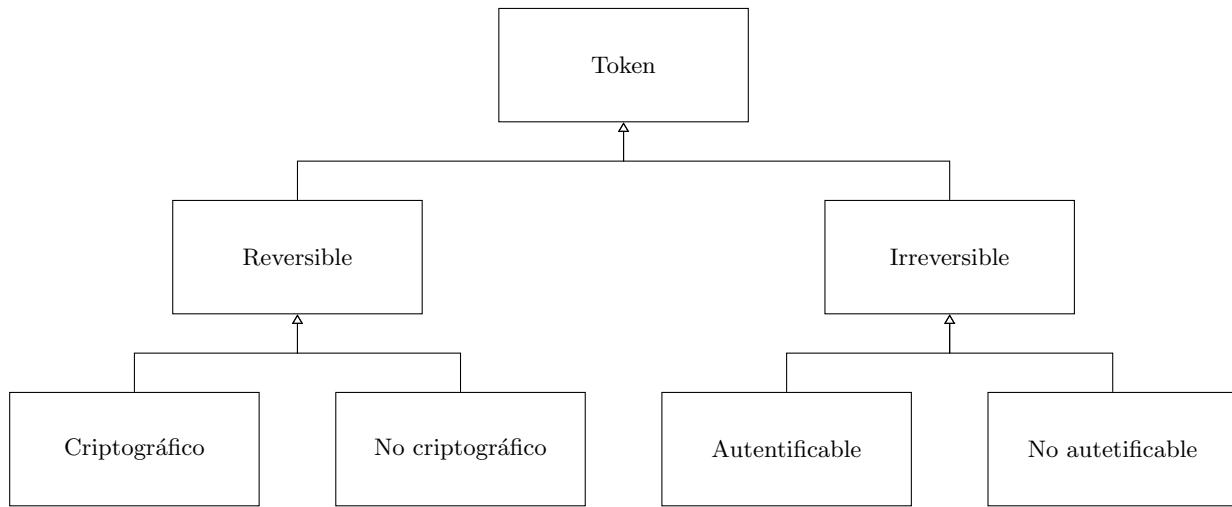


Figura 3.1: Clasificación de los TOKENS según PCI SSC.

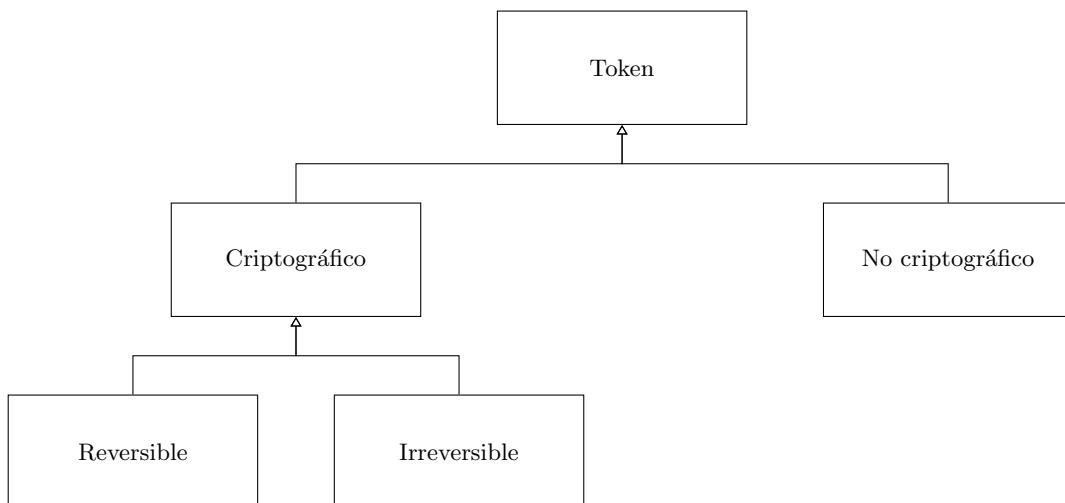


Figura 3.2: Clasificación propuesta de los TOKENS.

- TOKENS no criptográficos:
 - TKR (véase sección 3.2.3).
 - AHR (véase sección 3.2.4).
 - DETERMINISTIC RANDOM BIT GENERATOR (DRBG) (véase sección 3.2.5).

Cabe resaltar que, de acuerdo con esta clasificación, los TOKENS generados mediante DRBG pueden quedar también en la clasificación de TOKEN no reversible, no autenticable. Se propone una nueva clasificación, pues esta no parece muy acertada; por ejemplo, del lado de los irreversibles se pueden generar TOKENS autenticables (o no autenticables) mediante funciones hash criptográficas, sin embargo, estos TOKENS no son *criptográficos*. Respecto a los reversibles, se toman por ejemplo, a los algoritmos TKR, que hace uso de PRIMITIVAS CRIPTOGRÁFICAS, y AHR, que hace uso de un cifrado por bloques y una función hash criptográfica; ninguno de los cuales queda en la categoría de los criptográficos, pues se guardan las relaciones PERSONAL ACCOUNT NUMBER (PAN)-TOKEN en una base de datos.

Clasificación propuesta

- Criptográficos
 - Reversibles:
 - FFX (véase sección 3.2.1).
 - BPS (véase sección 3.2.2).
 - Irreversibles:
 - TKR (véase sección 3.2.3).
 - AHR (véase sección 3.2.4).
 - DRBG (véase sección 3.2.5).
- No criptográficos:
 - NON-DETERMINISTIC RANDOM BIT GENERATOR (NRBG).

La clasificación propuesta divide primeramente entre los que utilizan criptografía y los que no; se pone como ejemplo a los item NRBG como algoritmos no criptográficos, ya que su aleatoriedad depende de fenómenos físicos. Luego, dentro de los criptográficos, se divide a su vez entre algoritmos reversibles e irreversibles: los primeros, al igual que en el PCI SSC, son aquellos algoritmos que, dados el TOKEN y la llave, pueden regresar y obtener el PAN; los algoritmos pertenecientes a la segunda subcategoría (irreversible) son aquellos que necesitan guardar la relación PAN-TOKEN para poder obtener el PAN perteneciente a un TOKEN. Dado que, de acuerdo con el NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY (NIST), los DRBG utilizan funciones hash o cifrados por bloque, pertenecen también a la categoría de los criptográficos irreversibles.

3.2.1. Algoritmo FFX

FORMAT-PRESERVING FEISTEL-BASED ENCRYPTION (FFX) es un cifrado que preserva el formato presentado en [42] por Mihir Bellare, Phillip Rogaway y Terence Spies. En su forma más general, el algoritmo se compone de 9 parámetros que permiten cifrar cadenas de cualquier longitud en cualquier alfabeto; los autores también proponen dos formas más específicas (dos colecciones de parámetros) para alfabetos binarios y alfabetos decimales: A2 y A10, respectivamente. El NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY (NIST) llama a este método FF1 en su estándar sobre los cifrados que preservan el formato[29].

FFX ocupa redes Feistel (ver sección 2.2.3) junto con PRIMITIVAS CRIPTOGRÁFICAS adaptadas (usadas como función de ronda) para lograr preservar el formato. La idea general para las posibles funciones de ronda es interpretar la salida binaria de una primitiva (p. ej. un cifrado por bloques, una función hash, un cifrado de flujo) de forma que tenga el formato deseado; esto es, que tenga la misma longitud y se encuentre en el mismo alfabeto que la entrada. Es importante notar que esta función no tiene por qué ser invertible: las redes Feistel ocupan la misma operación tanto para cifrar como para descifrar.

La operación general del algoritmo es la misma que la operación de una red Feistel. Para redes desbalanceadas:

$$\begin{aligned} L_i &= R_{i-1} \\ R_i &= F_k(R_{i-1}) \oplus L_{i-1} \end{aligned} \tag{3.1}$$

Y para redes alternantes:

$$\begin{aligned} L_i &= \begin{cases} F_k^1(R_{i-1}) \oplus L_{i-1}, & \text{si } i \text{ es par} \\ L_{i-1}, & \text{si } i \text{ es impar} \end{cases} \\ R_i &= \begin{cases} R_{i-1}, & \text{si } i \text{ es par} \\ F_k^2(L_{i-1}) \oplus R_{i-1}, & \text{si } i \text{ es impar} \end{cases} \end{aligned} \tag{3.2}$$

Especificación de parámetros

A continuación se espifican los 9 parámetros de FFX.

1. **Radix.** Número que determina el alfabeto usado. $C = \{0, 1, \dots, \text{radix} - 1\}$. Tanto el texto en claro como el texto cifrado pertenecen a este alfabeto.
2. **Longitudes.** El rango permitido para longitudes de mensaje.
3. **Llaves.** El conjunto que representa al espacio de llaves.

Parámetro	Valor	Comentario
<i>radix</i>	10	alfabeto de dígitos decimales
longitudes	[4, 36]	rango de cadenas aceptadas
llaves	$\{0, 1\}^{128}$	misma longitud que para ADVANCED ENCRYPTION STANDARD (AES)
<i>tweaks</i>	$\text{BYTE}^{\leq 2^{64}-1}$	cadena de longitud arbitraria
suma	por bloque	combinación por operaciones a nivel de bloque
método	alternante	redes Feistel alternantes
<i>split</i>	0	lo más cercano al centro posible
rondas	12, 18 o 24	depende de longitud de mensaje

Tabla 3.1: Colección de parámetros FFX A10.

4. ***Tweaks***. El conjunto que representa al espacio de *tweaks*.
5. **Suma**. El operador utilizado en la red Feistel para combinar la parte izquierda con la salida de la función de ronda.
6. **Método**. El tipo de red Feistel a ocupar: desbalanceada o alternante.
7. ***Split***. El grado de desbalanceo de la red Feistel.
8. **Rondas**. El número de rondas de la red Feistel.
9. **F**. La función de ronda. Recibe la llave, el *tweak*, el número de ronda y un mensaje; regresa una cadena del alfabeto de la longitud apropiada.

FFX A10

De las dos colecciones de parámetros presentadas en [43], la que se adapta mejor al dominio de los números de tarjetas de crédito es A10, por lo que es la única que se presenta aquí. En la tabla 3.1 se muestran los valores con los que la colección FFX A10 inicializa FFX para crear un cifrado en el dominio decimal.

La dependencia entre el número de rondas y la longitud del mensaje está dada por la siguiente relación (*n* es la longitud del mensaje):

$$\text{rondas} = \begin{cases} 12 & \text{si } 10 \leq n \leq 36 \\ 18 & \text{si } 6 \leq n \leq 9 \\ 24 & \text{si } 4 \leq n \leq 5 \end{cases} \quad (3.3)$$

En el pseudocódigo 3.1 se describe a la función de ronda. Esta concatena el *tweak*, el mensaje de entrada y los demás parámetros usados por la red en una sola cadena; esta cadena se cifra con AES CIPHER-BLOCK CHAINING (CBC) MESSAGE AUTHENTICATION CODE (MAC); la salida se parte en dos

```

1 entrada: Arreglo número; llave k; tweak t; información adicional i
2 salida: Arreglo salida
3 inicio
4     entradaPrimitiva ← número || t || i
5     mac ← CBC_MAC_AES(entradaPrimitiva, k)
6     y' ← mac[1 ... 64]
7     y'' ← mac[65 ... 128]
8     si m ≤ 9 entonces:
9         z ← y'' mód 10m
10    sino:
11        z ← (y' mód 10m-9) × 109 + (y'' mód 10m)
12    fin
13    regresar z
14 fin

```

Pseudocódigo 3.1: *FFX A10*, función de ronda

(mitad derecha y mitad izquierda) y es operada para que el número resultado tenga el mismo número de dígitos que la cadena de entrada. Esta última operación se describe con la siguiente ecuación (y' es la parte izquierda y y'' la derecha):

$$z = \begin{cases} y'' \text{ mód } 10^m, & \text{si } m \leq 9 \\ (y' \text{ mód } 10^{m-9}) \cdot 10^9 + (y'' \text{ mód } 10^m), & \text{en cualquier otro caso} \end{cases} \quad (3.4)$$

El valor de m corresponde al *split* en la ronda actual; esto es la longitud de la cadena de entrada.

3.2.2. Algoritmo BPS

La información que aquí se presenta puede ser consultada con mayor detalle en [44].

BRIER-PEYRIN-STERN (BPS) es uno de los algoritmos de cifrado que preservan el formato existentes, y es capaz de cifrar cadenas de longitudes casi arbitrarias que estén formadas por cualquier conjunto de caracteres. El NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY (NIST) llama a este método FF3 en su estándar sobre los cifrados que preservan el formato[29].

BPS está conformado por 2 partes fundamentales, un cifrado interno *BC*, encargado de cifrar bloques de longitud fija; y un modo de operación, encargado de extender la funcionalidad de el cifrado *BC* y permitir que BPS cifre cadenas de varias longitudes.

El cifrado interno *BC*

El cifrado por bloques que usa BPS internamente se define como

$$BC_{F,s,b,w}(X, K, T) \quad (3.5)$$

Donde:

- F es un cifrado por bloques de f bits de salida, por ejemplo: TRIPLE DES (TDES), ADVANCED ENCRYPTION STANDARD (AES), SECURE HASH ALGORITHM (SHA)-2.
- s es la cardinalidad del conjunto de caracteres del bloque a cifrar.
- b es la longitud del bloque a cifrar, cumpliendo con $b \leq 2 \cdot |\log_s(2^{f-32})|$.
- w es el número (par) de rondas de la red Feistel interna (véase 2.2.3).
- X es la cadena o bloque de longitud b a cifrar.
- K es una llave acorde al cifrado por bloques F .
- T es un *tweak* de 64 bits.

Proceso de cifrado BC.

Para poder cifrar la cadena X :

1. Se tiene que dividir el *tweak* T de 64 bits en 2 *subtweaks* T_L y T_R de 32 bits. Viendo a T como un número entero codificado en binario se puede calcular $T_R = T \bmod 2^{32}$ y $T_L = (T - T_R)/2^{32}$
2. Igualmente, se tiene que dividir en 2 la cadena X para obtener las subcadenas X_L y X_R con una longitud l y r respectivamente. Dado que la longitud b de la cadena no siempre es par, se tiene que, si b es par, entonces tanto l como r son igual a $b/2$, pero en caso de que b sea impar, l va a ser igual a $(b + 1)/2$ y r igual a $(b - 1)/2$.
3. Partiendo de que el cifrado *BC* se compone de w rondas de una red Feistel, se define L_i y R_i (parte izquierda y parte derecha de la red en la i -ésima ronda), y se inicializan en:

$$L_0 = \sum_{j=0}^{l-1} X_L[j] \cdot s^j \quad (3.6)$$

$$R_0 = \sum_{j=0}^{r-1} X_R[j] \cdot s^j \quad (3.7)$$

```

1 entrada:     bloque  $N_w$  de longitud  $n$ .
2 salida:      bloque  $Y_N$ 
3 inicio
4   para  $i = 0$  hasta  $n - 1$ 
5      $Y_N[i] = N_w \text{ mod } s$ 
6      $N_w = (N_w - Y_N[i])/s$ 
7 fin

```

Pseudocódigo 3.2: Proceso de descomposición de L_w o R_w .

4. Ahora por cada ronda $i < w$ y procesando con el cifrado por bloques E .

Si i es par:

$$L_{i+1} = L_i \boxplus E_K((T_R \oplus i) \cdot 2^{f-32} + R_i) \quad (\text{mod } s^l) \quad (3.8)$$

$$R_{i+1} = R_i \quad (3.9)$$

Si i es impar:

$$R_{i+1} = R_i \boxplus E_K((T_L \oplus i) \cdot 2^{f-32} + L_i) \quad (\text{mod } s^r) \quad (3.10)$$

$$L_{i+1} = L_i \quad (3.11)$$

5. Por último se tiene que descomponer tanto a L_w como a R_w para obtener a Y_L y a Y_R respectivamente, las cuales concatenadas ($Y_L \parallel Y_R$) dan la cadena de salida Y .

El proceso para hacer la descomposición se muestra en el pseudocódigo 3.2.

De forma general, el proceso de cifrado se describe en el pseudocódigo 3.3.

Proceso de descifrado BC^{-1} .

Para poder descifrar la cadena Y :

1. Se tiene que dividir en 2 la cadena Y , para obtener las subcadenas Y_L y Y_R con una longitud l y r respectivamente, de igual forma que se hizo con la cadena X en el proceso de cifrado.
2. Partiendo de que el proceso de descifrado se compone de w rondas, se define L_i y R_i y se inicializan en:

$$L_w = \sum_{j=0}^{l-1} Y_L[j] \cdot s^j \quad (3.12)$$

$$R_w = \sum_{j=0}^{r-1} Y_R[j] \cdot s^j \quad (3.13)$$

```
1  entrada: la llave  $K$ ,  
2          el tweak  $T$ ,  
3          la cadena  $X$  de longitud  $b$  formada por el conjunto  $S$   
4          de cardinalidad  $s$ ,  
5          la función de cifrado  $F$ , y el número de rondas  $w$ .  
6  salida: La cadena cifrada  $Y$ .  
7  inicio  
8      calcular  $T_R = T \bmod 2^{32}$  y  $T_L = (T - T_R)/2^{32}$   
9      asignar  $l = \lceil b/2 \rceil$  y  $r = \lfloor b/2 \rfloor$   
10     inicializar  $L_0 = \sum_{j=0}^{l-1} X[j] \cdot s^j$   
11     inicializar  $R_0 = \sum_{j=0}^{r-1} X[j+l] \cdot s^j$   
12     para  $i = 0$  hasta  $i = w - 1$   
13         si  $i$  es par  
14              $L_{i+1} = L_i \boxplus F_K((T_R \oplus i) \cdot 2^{f-32} + R_i) \pmod{s^l}$   
15              $R_{i+1} = R_i$   
16         si  $i$  es impar  
17              $R_{i+1} = R_i \boxplus F_K((T_L \oplus i) \cdot 2^{f-32} + L_i) \pmod{s^r}$   
18              $L_{i+1} = L_i$   
19     para  $i = 0$  hasta  $i = l - 1$   
20          $Y_L[i] = L_w \bmod s$   
21          $L_w = (L_w - Y_L[i])/s$   
22     para  $i = l$  hasta  $i = r - 1$   
23          $Y_R[i] = R_w \bmod s$   
24          $R_w = (R_w - Y_R[i])/s$   
25     determinar  $Y = Y_L \parallel Y_R$   
26  fin
```

Pseudocódigo 3.3: Proceso de cifrado BC .

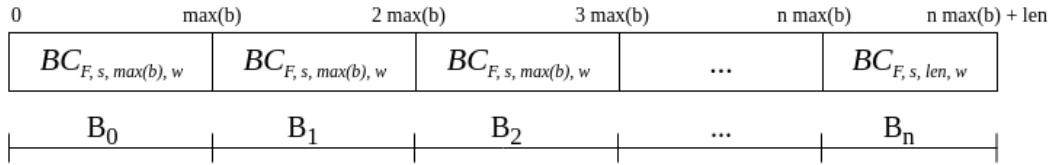


Figura 3.3: Corrimiento de cursor para la selección del último bloque en el modo de operación de BPS.

3. Ahora, comenzando con $i = w - 1$, para cada ronda $i \geq 0$.

Si i es par:

$$L_i = L_{i+1} \boxminus E_K((T_R \oplus i) \cdot 2^{f-32} + R_{i+1}) \quad (\text{mod } s^l) \quad (3.14)$$

$$R_i = R_{i+1} \quad (3.15)$$

Si i es impar:

$$R_i = R_{i+1} \boxminus E_K((T_L \oplus i) \cdot 2^{f-32} + L_{i+1}) \quad (\text{mod } s^r) \quad (3.16)$$

$$L_i = L_{i+1} \quad (3.17)$$

4. Finalmente se tienen que descomponer L_0 y R_0 (con el mismo proceso de descomposición descrito en el cifrado) para obtener a X_L y X_R , las cuales concatenadas ($X_L \parallel X_R$) dan la cadena de salida X .

De forma general, el proceso de descifrado se describe en el pseudocódigo 3.4.

El modo de operación

En cuanto al modo de operación de BPS, se puede decir que es un equivalente al modo de operación CIPHER-BLOCK CHAINING (CBC) (véase 2.2.6), ya que el bloque BC_n utiliza el texto cifrado de la salida del bloque BC_{n-1} , con la distinción de que en lugar de aplicar operaciones *xor* usa sumas modulares carácter por carácter, y de que no utiliza un VECTOR DE INICIALIZACIÓN, a pesar de soportar su uso.

Algo importante a resaltar de este modo de operación es que, en caso de que el texto en claro no tenga una longitud total que sea múltiplo de la longitud de bloque b , al cifrar el último bloque se recorre el cursor que determina el inicio del mismo, hasta que su longitud concuerde con b , esto se puede ver de forma gráfica en la figura 3.3.

En la figura 3.4 se ve de manera gráfica el funcionamiento del modo de operación de BPS.

Otra particularidad del modo de operación es el uso del contador u de 16 bits, que es utilizado para aplicar una operación *xor* al *tweak* T que entra a cada uno de los bloques BC . Recordando que T es de 64 bits, el *xor* se aplica a los 16 bits más significativos de ambas mitades de *tweak*, esto debido a que cada

```
1  entrada: la llave  $K$ ,  
2          el tweak  $T$ ,  
3          la cadena  $Y$  de longitud  $b$  formada por el conjunto  $S$   
4          de cardinalidad  $s$ ,  
5          la función de cifrado  $F$ ,  
6          el número de rondas  $w$ .  
7  salida: La cadena  $X$ .  
8  inicio  
9      calcular  $T_R = T \bmod 2^{32}$  y  $T_L = (T - T_R)/2^{32}$   
10     asignar  $l = \lceil b/2 \rceil$  y  $r = \lfloor b/2 \rfloor$   
11     inicializar  $L_w = \sum_{j=0}^{l-1} Y[j] \cdot s^j$   
12     inicializar  $R_w = \sum_{j=0}^{r-1} Y[j + l] \cdot s^j$   
13     para  $i = w - 1$  hasta  $i = 0$   
14       si  $i$  es par:  
15          $L_i = L_{i+1} \boxminus F_K((T_R \oplus i) \cdot 2^{f-32} + R_{i+1}) \pmod{s^l}$   
16          $R_i = R_{i+1}$   
17       si  $i$  es impar:  
18          $R_i = R_{i+1} \boxminus F_K((T_L \oplus i) \cdot 2^{f-32} + L_{i+1}) \pmod{s^r}$   
19          $L_i = L_{i+1}$   
20     para  $i = 0$  hasta  $i = l - 1$   
21        $X_L[i] = L_w \bmod s$   
22        $L_w = (L_w - X_L[i])/s$   
23     para  $i = l$  hasta  $i = r - 1$   
24        $X_R[i] = R_w \bmod s$   
25        $R_w = (R_w - X_R[i])/s$   
26     determinar  $X = X_L \parallel X_R$   
27  fin
```

Pseudocódigo 3.4: Proceso de descifrado BC^{-1} .

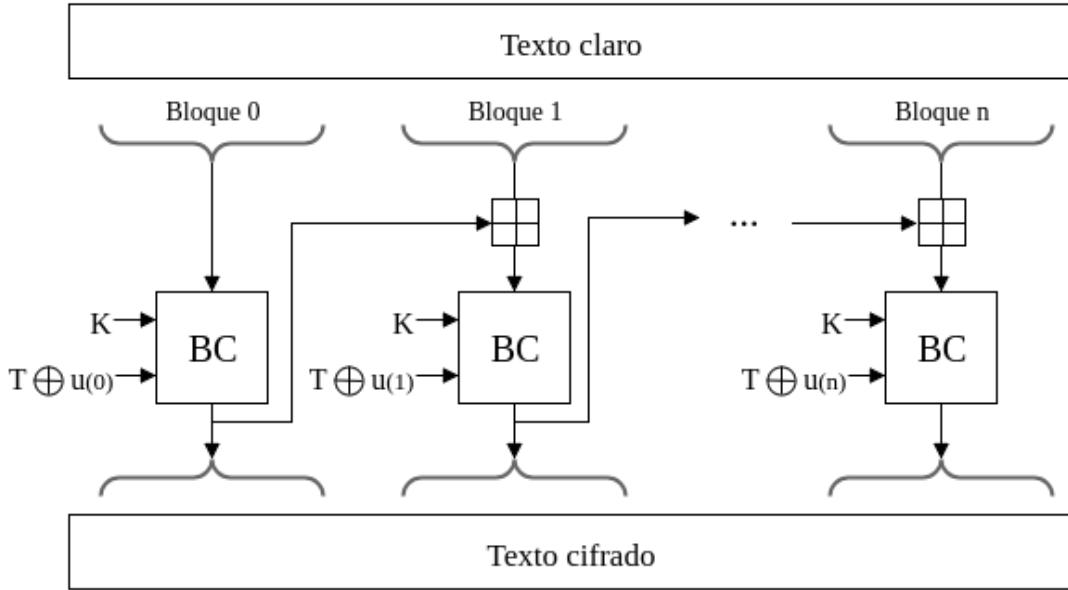


Figura 3.4: Modo de operación de BPS.

mitad de *tweak* funciona de manera independiente en el cifrado BC , y a que no se desea un traslape entre el contador externo u y el contador i interno en BC .

Características generales

Como se observó, BPS está basado en las redes Feistel, lo cual puede verse como una ventaja, debido al amplio estudio que tienen. Además, usa de forma interna algoritmos de cifrado o funciones hash estandarizadas, lo cual hace más comprensible y fácil su implementación.

BPS es un cifrado que preserva el formato capaz de cifrar cadenas de un longitud de 2 hasta $\max(b) \cdot 2^b$ caracteres (donde $\max(b)$ es el tamaño máximo de bloque), formadas por cualquier conjunto.

Se puede considerar que BPS es eficiente, debido a que la llave K usada en cada bloque BC es constante, y a que además usa un número reducido de operaciones internas en comparación con otros algoritmos de cifrado que preservan el formato.

Por último, se puede resaltar que el uso de *tweaks* protege a BPS de ataques de diccionario, los cuales son fáciles de cometer cuando el dominio de la cadena a cifrar es muy pequeño.

```
1 entrada: PAN p; información asociada d; llave k
2 salida: token
3 inicio
4      $S_1 \leftarrow \text{buscarPAN}(p)$ 
5      $S_2 \leftarrow \text{buscarInfoAsociada}(d)$ 
6     si  $S_1$  y  $S_2 = 0$ :
7          $t \leftarrow \text{RN}(k)$ 
8         insertar(t, p, d)
9     sino:
10         $t \leftarrow S_1$ 
11    fin
12    regresar t
13 fin
```

Pseudocódigo 3.5: *TKR2*, método de tokenización

Recomendaciones

Se recomienda que el número de rondas w de la red Feistel sea 8, dado que es un número de rondas eficiente, y se ha estudiado la seguridad de BPS con este w .

Es recomendable que como *tweak* se use la salida truncada de una función hash, en donde la entrada de la función puede ser cualquier información relacionada a los datos que se deseen proteger, como por ejemplo fechas, lugares, o parte de los datos que no se deseen cifrar.

3.2.3. Algoritmo TKR2

En [45] se analiza formalmente el problema de la generación de TOKENS y se propone un algoritmo que no está basado en FORMAT PRESERVING ENCRYPTION (FPE). Hasta antes de la publicación de este documento, los únicos métodos para generar TOKENS cuya seguridad estaba formalmente demostrada eran los basados en FPE.

El algoritmo propuesto usa PRIMITIVAS CRIPTOGRÁFICAS para generar TOKENS aleatorios y almacena en una base de datos (CARD DATA VAULT (CDV)) la relación original de estos con los PERSONAL ACCOUNT NUMBER (PAN). Según la clasificación del PAYMENT CARD INDUSTRY (PCI) se trata de una algoritmo tokenizador reversible *no criptográfico*; sin embargo tratarlo de esta forma (*como no criptográfico*) resulta un tanto confuso, dado que toda la contrucción del algoritmo y las pruebas de su seguridad se basan en fundamentos de la criptografía.

En el pseudocódigo 3.5 se muestra el proceso de tokenización, mientras que en 3.6 está la detokenización.

```

1 entrada: token t; información asociada d; llave k
2 salida: PAN
3 inicio
4      $S_1 \leftarrow \text{buscarToken}(t)$ 
5      $S_2 \leftarrow \text{buscarInfoAsociada}(d)$ 
6     si  $S_1$  y  $S_2 = 0$ :
7         regresar error
8     sino:
9         p  $\leftarrow S_1$ 
10    fin
11    regresar p
12 fin

```

Pseudocódigo 3.6: *TKR2*, método de detokenización

La mayor parte del proceso de tokenización y toda la detokenización son bastante fáciles de comprender; lo único que queda por esclarecer es la función generadora de TOKENS aleatorios RN_k . Idealmente, esta función debe regresar un elemento uniformemente aleatorio del espacio de TOKENS. La propuesta que se hace en [45] para instanciar esta función se presenta en el pseudocódigo 3.7. Aquí, la variable *contador* mantiene un estado del algoritmo (mantiene su valor a lo largo de las distintas llamadas); el espacio de TOKENS contiene cadenas de longitud fija μ de un alfabeto AL cuya cardinalidad es l ; el número de bits necesarios para enumerar a todo el alfabeto se guardan en $\lambda = \lceil \log_2 l \rceil$.

En el pseudocódigo 3.7 lo primero que se hace es utilizar una PRIMITIVA CRIPTOGRÁFICA f para generar una cadena binaria (hablaremos sobre este punto más adelante); esta cadena (nombrada X) se parte en subcadenas de λ bits; después se itera de manera consecutiva sobre estas subcadenas binarias, si la representación entera de la i -ésima está en el rango del alfabeto de los TOKENS, entonces se concatena al token resultado, sino, se pasa a la siguiente subcadena. La longitud de la cadena regresada por f debe ser, aproximadamente, $3\mu\lambda$: dado que se espera que el comportamiento de f sea EQUIPROBABLE, entonces el ciclo correrá un promedio de 2^μ veces.

Existen varios candidatos viables para f : un cifrado de flujo, pues el flujo de llave de estos produce cadenas de aspecto aleatorio; un cifrado por bloques (sección 2.2), utilizando un modo de operación de contador (sección 2.2.6); un TRUE RANDOM NUMBER GENERATOR (TRNG) para obtener secuencias de bits verdaderamente aleatorias.

Por último hay que aclarar que el algoritmo presentado en el pseudocódigo 3.5 debe recibir un par de modificaciones más: al momento de generar un TOKEN debe existir una validación que verifique que este sea único (para evitar que dos PAN tengan un mismo TOKEN); la base de datos debe estar cifrada, por lo que, antes de hacer inserciones y después de hacer consultas, deben existir las operaciones correspondientes.

```
1  entrada: llave k
2  salida: token
3  inicio
4       $X \leftarrow f(k, \text{ contador})$ 
5       $X_1, X_2, \dots, X_m \leftarrow \text{cortar}(X, \lambda)$ 
6      t  $\leftarrow ""$ 
7      i  $\leftarrow 0$ 
8      mientras  $|t| \neq \mu:$ 
9          si entero( $X_{-i}$ )  $\leq l:$ 
10             t  $\leftarrow t + \text{entero}(X_{-i})$ 
11         fin
12         i  $\leftarrow i + 1$ 
13     fin
14     contador  $\leftarrow$  contador + 1
15     regresar t
16 fin
```

Pseudocódigo 3.7: *TKR2*, generación de TOKENS aleatorios

3.2.4. Algoritmo Híbrido Reversible

Longo, Aragona y Sala [46], propusieron en 2017, un algoritmo de tipo híbrido reversible. Este está basado en un cifrado de bloques con una llave secreta y una entrada adicional.

Se sabe que el número de una tarjeta (PERSONAL ACCOUNT NUMBER (PAN)) está conformado por tres partes concatenadas: el número que identifica al emisor de la tarjeta, el que identifica la cuenta y un número de verificación. En este algoritmo se reemplaza la primera parte con un TOKEN BANK IDENTIFIER NUMBER (BIN) y se cifra solo la parte que identifica la cuenta. Al final, se calcula un nuevo dígito de verificación.

Las entradas del algoritmo son la parte del PAN a cifrar y una entrada adicional. Esta última actúa como un *tweak* (véase sección ??), pues permite que se generen varios TOKENS para el mismo PAN.

El algoritmo necesita una función f pública que, dada una cadena de longitud m regrese una de longitud n (véase sección de funciones hash 2.3). Se toman solo cifrados cuyo tamaño de bloque sea de mínimo 128 bits. La función f se encarga de poner el relleno en la entrada para completar el bloque del cifrado y permitir la creación de varios TOKENS para el mismo PAN utilizando la misma llave en el proceso de cifrado. Finalmente, el algoritmo necesita una base de datos segura que se encargará de contener los pares PAN-TOKEN. Al momento de crear los TOKENS, se necesita acceder a la base de datos mediante una FUNCIÓN BOLEANA *comprobar* que revisa si el TOKEN generado ya está almacenado en la base.

Como se desea obtener un TOKEN que tenga el mismo número de dígitos que el PAN (longitud l)

```
1  entrada: PAN p; entrada_adicional u; llave k
2  salida: token
3  inicio
4       $t = f(u, p) \parallel [\bar{p}]_b^s$  (paso 1)
5       $c = E(k, t)$  (paso 2)
6      si ( $\bar{c}$  mód  $2^n$ )  $\geq 10^l$ 
7           $t = c$ 
8          Regresar al paso 2.
9      fin
10      $token = [\bar{c}$  mód  $2^n]_{10}^l$ 
11     si comprobar(token) = verdadero
12          $u = u + 1$ 
13         Regresar al paso 1.
14     fin
15     regresar token
16 fin
```

Pseudocódigo 3.8: Híbrido reversible, método de tokenización

ingresado, se deben tomar en cuenta solo una fracción de las posibles salidas del cifrado E ; para resolver este problema, se utiliza un método conocido como el CIFRADO DE CAMINATA CÍCLICA.

Notación

A continuación se definen una serie de notaciones que se utilizarán en el algoritmo:

- M Tamaño de bloque del cifrado por bloques que se usará.
- l Longitud de la entrada. En este caso, $13 \geq l \geq 19$.
- n Número de bits necesarios para representar a la entrada: $n = \log_2(10^l)$.
- $[y]_b^s$ Indica que y es menor que b^s : $y < b^s$.
- \bar{x} Representación de x en una cadena binaria cuando x es representado en su forma decimal y viceversa.

El pseudocódigo del algoritmo de tokenización se puede observar en ??., ,

```
1 entrada: número n de bytes deseados .
2 salida: arreglo de n bytes .
3 inicio
4     longitud  $\leftarrow$  tamanioDeHash ()
5     numeroDeBloques  $\leftarrow$   $\lceil \text{longitud}/n \rceil$ 
6     datos  $\leftarrow$  semilla
7     para_todo i en numeroDeBloques :
8         resultado  $\mid\mid=$  hash (datos)
9         datos  $\doteqdot=$  1
10    fin
11    regresar n bytes de resultado
12 fin
```

Pseudocódigo 3.9: Generación de bits pseudoaleatorios mediante función hash

3.2.5. Tokenización mediante generador de números pseudoaleatorios

En la sección B se habló sobre el estándar del NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY (NIST) para la generación de números aleatorios. En esta mostramos cómo estos métodos pueden ser utilizadas para tokenizar y detokenizar números de tarjetas. Primero se muestran dos posibles instancias con PRIMITIVAS CRIPTOGRÁFICAS del método general descrito en B: una basada en funciones hash (sección 2.3) y otra basada en cifrados por bloque (sección 2.2); ambas presentadas en [34].

DRBG basado en funciones hash

El método genérico define un solo valor crítico (un valor que se debe mantener en secreto, pues la seguridad en generador depende de esto): la semilla, tanto al inicio, como su valor a lo largo de su vida útil. Este método introduce un segundo valor crítico: una constante *C* cuyo valor depende también de la entrada de entropía. Por tanto, las funciones de instancia, cambio de semilla y desinstancia se deben modificar para incluir a este segundo valor.

En el pseudocódigo 3.9 se muestra la función de generación de bits pseudoaleatorios basada en una función hash. *tamanioDeHash* regresa el tamaño de los *digest* retornados por la función hash usada.

```
1 entrada: número  $n$  de bytes deseados .
2 salida: arreglo de  $n$  bytes .
3 inicio
4   mientras longitud (resultado)  $\leq n$ :
5     semilla += 1
6     resultado ||= cifrar (semilla)
7   fin
8   regresar  $n$  bytes de resultado
9 fin
```

Pseudocódigo 3.10: Generación de bits pseudoaleatorios mediante cifrado por bloques

DRBG basado en cifrado por bloque

Al igual que el generador basado en funciones hash, este método también introduce un segundo valor crítico: el valor de la llave utilizada por el cifrado por bloque subyacente. Esta llave se genera a partir de la entrada de entropía y debe ser tratada de igual forma que la semilla por las funciones de instanciación, cambio de semilla y desinstanciación.

En el pseudocódigo 3.10 se muestra la función de generación de bits psudoaleatorios basada en un cifrado por bloques.

Uso como algoritmo tokenizador

Un DETERMINISTIC RANDOM BIT GENERATOR (DRBG) puede ser utilizado de manera bastante similar a la función RN del algoritmo TKR (sección 3.2.3): en lugar de utilizar la función f como método para obtener bits pseudoaleatorios, se ocupa el generador pseudoaleatorio. El pseudocódigo para esto se muestra en 3.11; la función $drbg$ es una llamada a la operación de generación de bits pseudoaleatorios de cualquiera de los dos generadores presentados anteriormente.

Al igual que con TKR, ambas operaciones (tokenización y detokenización) deben tener el soporte de una base de datos: al generar un nuevo TOKEN este se guarda en la base de datos, y la operación de detokenización es solamente una consulta en la base.

```
1  entrada: llave k
2  salida: token
3  inicio
4       $X \leftarrow \text{drbg}()$ 
5       $X_1, X_2, \dots, X_m \leftarrow \text{cortar}(X, \lambda)$ 
6      t  $\leftarrow \text{"")}$ 
7      i  $\leftarrow 0$ 
8      mientras  $|t| \neq \mu:$ 
9          si entero( $X_{-i}$ )  $\leq l:$ 
10             t  $\leftarrow t + \text{entero}(X_{-i})$ 
11         fin
12         i  $\leftarrow i + 1$ 
13     fin
14     contador  $\leftarrow \text{contador} + 1$ 
15     regresar t
16 fin
```

Pseudocódigo 3.11: Generación de *tokens* mediante DRBG

Capítulo 4

Análisis y diseño de programa tokenizador

*«A common mistake that people make
when trying to design something
completely foolproof is to underestimate
the ingenuity of complete fools.»*

Mostly Harmless, DOUGLAS ADAMS.

La metodología ocupada en el proyecto es una mezcla entre prototipos y SECURITY DEVELOPMENT LIFE CYCLE (SDL). En este capítulo se abordan los procesos de análisis y diseño del primer prototipo del proyecto: el módulo de algoritmos generadores de TOKENS.

4.1. Requerimientos

El PAYMENT CARD INDUSTRY (PCI) DATA SECURITY STANDARD (DSS) define cuatro *dominios* de seguridad para el proceso de tokenización:

1. **Generación de TOKENS.** Para cada clase tokenizadora, este dominio se encarga de definir consideraciones para la generación segura de TOKENS. Cubre los dispositivos, procesos, mecanismos y algoritmos que son utilizados para crear los TOKENS.
2. **Mapeo de TOKENS.** Este dominio, que se refiere al mapeo de los TOKENS con su PERSONAL ACCOUNT NUMBER (PAN) origen, aplica solamente a los procesos de tokenización reversibles. Entre otras cosas, provee guías respecto a control de acceso necesarios para las peticiones de tokenización.
3. **Bóveda de datos de tarjeta.** Como el dominio pasado, solo aplica a las implementaciones de tokenización reversibles. Cubre el cifrado obligado del PAN y los controles de acceso necesarios para entrar a la CARD DATA VAULT (CDV).
4. **Manejo criptográfico de llaves.** Define las buenas prácticas para el manejo criptográfico de las llaves y las operaciones realizadas con ellas por el producto tokenizador.

En esta sección se detallan los requerimientos que deben de satisfacer los TOKENS según las guías de tokenización del PCI DSS [40]. Para comenzar, se enlistan los requerimientos aplicables a todos los TOKENS, sin importar su categoría; después de esto, se agrupan los requerimientos correspondientes a cada una de las categorías presentadas en la sección 3.1.

Para su posterior seguimiento, cada requerimiento se clasifica según la entidad responsable de su cumplimiento; estas categorías son:

1. **Algoritmos.** Requerimiento cuyo cumplimiento es responsabilidad del diseñador de un algoritmo tokenizador.
2. **Implementaciones.** Requerimiento cuyo cumplimiento es responsabilidad del implementador de un algoritmo tokenizador.
3. **Sistema tokenizador.** Requerimiento cuyo cumplimiento es responsabilidad de un proveedor de servicios de tokenización.

Los requerimientos de la primera categoría se encuentran fuera de los propósitos de este trabajo. En las categorías dos y tres se agrupan los requerimientos que, en efecto, un trabajo como este debe de cumplir. A

los de la segunda categoría se les da seguimiento en la sección 4.1.4; los de la tercera son parte del siguiente módulo de este proyecto y se les da seguimiento en la sección 6.1.7.

REQPCI-01 Validación de productos de hardware.

Clasificación: SISTEMA TOKENIZADOR.

Si se usa un producto de hardware para la tokenización, este debe de ser validado por FEDERAL INFORMATION PROCESSING STANDARD (FIPS) 140-2 nivel 3 o superior (descrito en [47]).

REQPCI-02 Validación de productos de software.

Clasificación: SISTEMA TOKENIZADOR.

Si se usa un producto de software para la tokenización, este debe de ser validado por FIPS 140-2 nivel 2 o superior (descrito en [47]).

REQPCI-03 Resistencia a texto claro conocido.

Clasificación: ALGORITMOS.

Un atacante con acceso a múltiples pares de TOKENS y PAN no debe de ser capaz de determinar otros PAN a partir de solamente TOKENS. En otras palabras, los TOKENS deben ser resistentes a ataques con texto en claro conocido (sección 2.1.1).

REQPCI-04 Resistencia a sólo texto cifrado.

Clasificación: ALGORITMOS.

Recuperar un PAN a partir de un TOKEN debe de ser COMPUTACIONALMENTE NO FACTIBLE (resistencia a ataques con sólo texto cifrado, sección 2.1.1).

REQPCI-05 Detección de anomalías.

Clasificación: SISTEMA TOKENIZADOR.

Se deben de implementar disparadores que permitan detectar irregularidades en el sistema (anomalías, funcionamientos erróneos, comportamientos sospechosos). El producto debe registrar dichos eventos y avisar al personal correspondiente.

REQPCI-06 Distinción entre TOKENS y PAN.

Clasificación: IMPLEMENTACIONES.

Se debe contar con un mecanismo para distinguir entre TOKENS y PAN. Los proveedores del servicio de tokenización deben compartir este mecanismo con la entidad (o entidades) que usa los TOKENS.

REQPCI-07 Guía de instalación.

Clasificación: SISTEMA TOKENIZADOR.

Se debe de contar con una guía de instalación y uso para el correcto funcionamiento del producto de tokenización.

REQFPCI-08 Integridad del proceso de tokenización.

Clasificación: IMPLEMENTACIONES.

Deben de implementarse mecanismos que garanticen la integridad del proceso de generación de TOKENS.

REQFPCI-09 Acceso al proceso detokenización.

Clasificación: SISTEMA TOKENIZADOR.

Solo los usuarios autenticados y componentes del sistema tienen permitido acceder al proceso de tokenización y detokenización. Los métodos utilizados deben ser al menos tan rigurosos como lo indicado en el requerimiento 8 del PCI DSS [41].

SUBREQFPCI-09/1 Control de peticiones.

Todas las peticiones deben pasar a través de una APPLICATION PROGRAM INTERFACE (API) que controle todos los intentos de acceso y aplique de manera uniforme reglas de control de acceso.

SUBREQFPCI-09/2 Registros de acceso.

Se deben registrar todos los eventos de acceso, de tokenización y de detokenización. Esta funcionalidad debe ser configurable de manera segura. Para esto se debe seguir el requerimiento 4 del PAYMENT APPLICATION (PA)-DSS [48].

SUBREQFPCI-09/3 AUTENTICACIÓN MULTIFACTOR.

El sistema debe soportar AUTENTICACIÓN MULTIFACTOR para todos los tipos de usuario: accesos administrativos, operaciones de tokenización y detokenización, mantenimiento, etcétera.

SUBREQFPCI-09/4 Accesos a nivel de sistema.

Todos los accesos a nivel de sistema deben soportar AUTENTICACIÓN MUTUA, incluyendo a las peticiones de tokenización y detokenización.

SUBREQFPCI-09/5 Accesos administrativos.

Se debe utilizar CRIPTOGRAFÍA FUERTE para todos los accesos administrativos que no se hagan desde consola.

REQFPCI-10 Mapeos de TOKEN a TOKEN prohibidos.

Clasificación: IMPLEMENTACIONES.

No se debe poder pasar de un primer TOKEN válido a un segundo, también válido; forzosamente debe existir un estado intermedio: del primer TOKEN se pasa al PAN correspondiente (operación de detokenización) y de este se pasa al segundo TOKEN.

REQFPCI-11 Protección contra vulnerabilidades comunes.

Clasificación: IMPLEMENTACIONES.

Se deben implementar medidas en contra de las vulnerabilidades de seguridad más comunes ([48], requerimiento 5.2). Algunas de estas medidas pueden ser el uso de herramientas de análisis de código estático, o el uso de lenguajes de programación especializados.

REQFPCI-12 Primitivas criptográficas usadas.

Clasificación: IMPLEMENTACIONES.

Las primitivas criptográficas que se usen deben estar basadas en estándares nacionales (referentes a Estados Unidos) o internacionales (p. ej. ADVANCED ENCRYPTION STANDARD (AES)). Ver sección A.

REQFPCI-13 Sobre el manejo adecuado de llaves.

Clasificación: SISTEMA TOKENIZADOR.

En donde se usen llaves para la generación y protección de TOKEN, se deben seguir buenas prácticas criptográficas para la administración de estas. En particular, se deben cumplir con las recomendaciones del NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY (NIST) en [38] y [39].

SUBREQFPCI-13/1 Sobre el ciclo de vida.

La llave tokenizadora debe seguir la política de los ciclos de llaves descritos en el ISO/IEC 115681 (ver sección E).

SUBREQFPCI-13/2 Descripción del periodo criptográfico activo.

La política sobre el tiempo de vida de la llave debe incluir una descripción sobre el periodo criptográfico activo de la llave tokenizadora en cuestión.

SUBREQFPCI-13/3 Sobre la destrucción de las llaves.

El proveedor debe incorporar una función que permita la destrucción de sus llaves criptográficas sin tener que alterar o abrir el dispositivo.

SUBREQFPCI-13/4 Exportar llave en claro prohibido.

Las llaves usadas para generar TOKENS no se deben poder exportar en claro desde el programa.

SUBREQFPCI-13/5 Entropía de generación de llaves.

La fuente generadora de llaves debe tener, al menos, 128 bits de ENTROPÍA.

SUBREQPCI-13/6 Llaves de uso único.

Las llaves criptográficas usadas para generar TOKENS no deben ser usadas para ningún otro fin.

4.1.1. Irreversibles

REQPCI-14 Sobre la generación de TOKENS (irreversibles).

Clasificación: ALGORITMOS.

El mecanismo utilizado para la generación de los TOKENS no es reversible (o improbable).

SUBREQPCI-14/1 Sobre el mecanismo generador (irreversibles).

El proceso para crear TOKENS clasificado como irreversible debe asegurar que el mecanismo, proceso o algoritmo utilizado para crear el TOKEN no sea reversible. Si una función hash (véase sección 2.3) es utilizada, esta debe ser una PRIMITIVA CRIPTOGRÁFICA y utilizar una llave secreta k tal que el mero conocimiento de la función hash no permita la creación de un ORÁCULO.

SUBREQPCI-14/2 Contenido en claro (irreversibles).

Los TOKENS irreversibles no deben contener dígitos en claro del PERSONAL ACCOUNT NUMBER (PAN) original, excepto que estos dígitos sean una coincidencia.

SUBREQPCI-14/3 Creación de un diccionario (irreversibles).

La creación de una tabla o *diccionario* de TOKENS estáticos debería ser imposible, o, al menos, al punto de satisfacer que la probabilidad de predecir correctamente el PAN sea menor que $\frac{1}{10^6}$.

SUBREQPCI-14/4 Sobre el proceso de autenticación (irreversibles).

En el caso de los TOKENS autenticables, el proceso de autenticación no debe revelar información suficiente para realizar búsquedas, excepto una exhaustiva (PAN por PAN) y se deben implementar controles para detectar estas últimas.

4.1.2. Criptográficos reversibles

REQPCI-15 Probabilidad de adivinar relaciones (criptográficos).

Clasificación: ALGORITMOS.

La probabilidad de adivinar la relación entre un TOKEN y un PERSONAL ACCOUNT NUMBER (PAN) debe de ser menor que 1 en 10^6 .

SUBREQPCI-15/1 Distribución uniforme (criptográficos).

Para un PAN dado, todos los TOKENS deben ser equiprobables; esto es, el mecanismo tokenizador no debe exhibir tendencias probabilísticas que lo expongan a ataques estadísticos.

SUBREQFPCI-15/2 Permutación aleatoria (criptográficos).

El método de tokenización debe actuar como una familia de PERMUTACIONES aleatoria desde el espacio de PAN al espacio de TOKENS.

SUBREQFPCI-15/3 Cambio de llave (criptográficos).

Un cambio en la llave se debe ver reflejado en un cambio en el TOKEN resultado.

SUBREQFPCI-15/4 Cambio de PAN (criptográficos).

Un cambio en el PAN se debe ver reflejado en un cambio en el TOKEN resultado.

SUBREQFPCI-15/5 Verificación de la aleatoriedad (criptográficos).

Se debe tener un medio para verificar de forma práctica la aleatorización de dígitos, de acuerdo a lo establecido en NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY (NIST) 800-90A [34].

REQFPCI-16 Almacenamiento de TOKENS (criptográficos).

Clasificación: SISTEMA TOKENIZADOR.

Los TOKENS generados no se deben almacenar en ningún punto del sistema.

El requerimiento anterior (REQFPCI-16 ALMACENAMIENTO DE TOKENS (CRIPTOGRÁFICOS), RC1C en [40]) es un tanto difícil de interpretar; la versión original establece: *los TOKENS basados en el PAN completo no se deben almacenar si el producto tokenizador también almacena su PAN truncado correspondiente*. Es un requerimiento de los criptográficos reversibles, por lo que el TOKEN no se debería almacenar bajo ninguna circunstancia (según la propia clasificación del PAYMENT CARD INDUSTRY (PCI) SECURITY STANDARD COUNCIL (SSC)); la redacción del requerimiento se cambió para reflejar este hecho.

REQFPCI-17 Seguridad de la administración de llaves (criptográficos).

Clasificación: SISTEMA TOKENIZADOR.

Todas las operaciones sobre la administración de las llaves criptográficas deben realizarse en un dispositivo criptográfico seguro y aprobado: el PCI SSC se encarga de hacer validaciones; también puede ser cualquier dispositivo validado por FEDERAL INFORMATION PROCESSING STANDARD (FIPS) 140-2 nivel 3 o superior [47] o por la INTERNATIONAL ORGANIZATION FOR STANDARDIZATION (ISO) 13491-1.

REQFPCI-18 Sobre la longitud de las llaves (criptográficos).

Clasificación: IMPLEMENTACIONES.

Las llaves para *tokenizar* deben tener una FUERZA EFECTIVA de, al menos, 128 bits. Cualquier llave utilizada para proteger o para derivar la llave del TOKEN debe de ser de igual o mayor FUERZA EFECTIVA.

REQFPCI-19 Independencia estadística (criptográficos).

Clasificación: ALGORITMOS.

Si el espacio de llaves es usado para producir TOKENS en dos contextos distintos (p. ej. para distintos comerciantes), estas deben ser ESTADÍSTICAMENTE INDEPENDIENTES.

4.1.3. No criptográficos reversibles

REQFPCI-20 Generación y almacenamiento de tokens (no criptográficos).

Clasificación: ALGORITMOS.

La generación de un TOKEN debe realizarse independientemente de su PERSONAL ACCOUNT NUMBER (PAN), y la relación entre un PAN y su TOKEN sólo tiene que estar almacenada en la base de datos (CARD DATA VAULT (CDV)) establecida.

REQFPCI-21 Probabilidad de encontrar un PAN (no criptográficos).

Clasificación: ALGORITMOS.

La probabilidad de encontrar un PAN a partir de su respectivo TOKEN debe de ser menor que $1 \text{ en } 10^6$.

SUBREQFPCI-21/1 Distribución equiprobable (no criptográficos).

Para un PAN dado, todos sus TOKENS respectivos deben ser EQUIPROBABLES, esto es que el sistema *tokenizador* no debe exhibir patrones probabilísticos que lo vulneren a un ataque estadístico.

SUBREQFPCI-21/2 Permutaciones aleatorias (no criptográficos).

El método de tokenización debe actuar como una familia de PERMUTACIONES aleatoria en el espacio efectivo de los PANs al espacio de TOKENS.

SUBREQFPCI-21/3 Parámetros de tokenización (no criptográficos).

El método de tokenización debe incluir parámetros tales que, un cambio en estos parámetros resulte en un TOKEN diferente; por ejemplo, un cambio en la instancia del proceso debe derivar en una secuencia de TOKENS distintos, incluso cuando es usada la misma secuencia de TOKENS.

SUBREQFPCI-21/4 Verificación de la aleatoriedad (no criptográficos).

Se debe tener un medio para verificar de forma práctica la aleatorización de dígitos, de acuerdo a lo establecido en NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY (NIST) 800-90A [34].

En [40] se establece un subrequerimiento más de REQFPCI-21 PROBABILIDAD DE ENCONTRAR UN PAN (NO CRIPTOGRÁFICOS): *Al cambiar parte de un PAN, debe cambiar su TOKEN resultante.* Es un requerimiento análogo a SUBREQFPCI-15/4 CAMBIO DE PAN (CRIPTOGRÁFICOS), sin embargo en el contexto de los no criptográficos reversibles, tal restricción no tiene sentido, dado que la generación del TOKEN es independiente del PAN (requerimiento REQFPCI-20 GENERACIÓN Y ALMACENAMIENTO DE TOKENS (NO CRIPTOGRÁFICOS)).

REQFPCI-22 Distribución imparcial (no criptográficos).

Clasificación: ALGORITMOS.

El proceso de generación de TOKENS debe garantizar una distribución de TOKENS imparcial, esto significa que la probabilidad de cualquier par PAN/TOKEN debe ser igual.

REQFPCI-23 Instancias estadísticamente independientes (no criptográficos).

Clasificación: SISTEMA TOKENIZADOR.

Si varias o diferentes instancias de la bases de datos (CDV) son usadas, cada una de estas debe ser ESTADÍSTICAMENTE INDEPENDIENTES.

REQFPCI-24 Proceso de detokenización (no criptográficos).

Clasificación: ALGORITMOS.

El proceso de detokenización debe realizarse por medio de una búsqueda de datos o un índice dentro de la base de datos (CDV), y no por medio de métodos criptográficos.

Un subrequerimiento de REQFPCI-24 PROCESO DE DETOKENIZACIÓN (NO CRIPTOGRÁFICOS) que aquí se omite establece: «*El PAN y el TOKEN debe ser probabilísticamente independientes. Cualquier método lógico o matemático no debe ser usado para tokenizar el PAN o detokenizar el TOKEN.*» La independencia entre PAN y TOKEN ya se establece en REQFPCI-20 GENERACIÓN Y ALMACENAMIENTO DE TOKENS (NO CRIPTOGRÁFICOS). No es clara la ascepción de *método lógico matemático*; una solución común para generar TOKENS no criptográficos es usar PSEUDORANDOM NUMBER GENERATOR (PRNG), los cuales son métodos matemáticos.

REQFPCI-25 Cifrado de la base de datos (no criptográficos).

Clasificación: IMPLEMENTACIONES.

Dentro de la base de datos (CDV), los PAN deben ser cifrados con una llave de mínimo 128 bits de FUERZA EFECTIVA.

REQFPCI-26 Seguridad de la administración de llaves (no criptográficos).

Clasificación: SISTEMA TOKENIZADOR.

Todas las operaciones sobre la administración de las llaves criptográficas deben realizarse en un dispositivo criptográfico seguro y aprobado: el PAYMENT CARD INDUSTRY (PCI) SECURITY STANDARD COUNCIL (SSC) se encarga de hacer validaciones; también puede ser cualquier dispositivo validado por FEDERAL INFORMATION PROCESSING STANDARD (FIPS) 140-2 nivel 3 o superior [47] o por la INTERNATIONAL ORGANIZATION FOR STANDARDIZATION (ISO) 13491-1.

Los siguientes requerimientos son referentes al cumplimiento de distintos estándares y recomendaciones (principalmente del NIST); en la sección 10 se resume el contenido de cada uno de estos:

- REQFPCI-01 VALIDACIÓN DE PRODUCTOS DE HARDWARE.
- REQFPCI-02 VALIDACIÓN DE PRODUCTOS DE SOFTWARE.
- REQFPCI-09 ACCESO AL PROCESO DETOKENIZACIÓN.
- REQFPCI-13 SOBRE EL MANEJO ADECUADO DE LLAVES.
- SUBREQFPCI-15/5 VERIFICACIÓN DE LA ALEATORIEDAD (CRPTOGRÁFICOS).

La tabla 4.1 es una relación entre la lista de requerimientos aquí presentada y la notación del PCI SSC en [40]. Sobre todo en cuanto a los requerimientos REQFPCI-09 ACCESO AL PROCESO DETOKENIZACIÓN y REQFPCI-13 SOBRE EL MANEJO ADECUADO DE LLAVES el documento del PCI es bastante repetitivo: se colocan 3 versiones (una para cada categoría) con prácticamente el mismo contenido.

Requerimiento	Equivalente PCI
REQFPCI-01 VALIDACIÓN DE PRODUCTOS DE HARDWARE	GT1
REQFPCI-02 VALIDACIÓN DE PRODUCTOS DE SOFTWARE	GT3
REQFPCI-03 RESISTENCIA A TEXTO CLARO CONOCIDO	GT4
REQFPCI-04 RESISTENCIA A SÓLO TEXTO CIFRADO	GT5
REQFPCI-05 DETECCIÓN DE ANOMALÍAS	GT6
REQFPCI-06 DISTINCIÓN ENTRE TOKENS Y PAN	GT7
REQFPCI-07 GUÍA DE INSTALACIÓN	GT8
REQFPCI-08 INTEGRIDAD DEL PROCESO DE TOKENIZACIÓN	GT9
SUBREQFPCI-09/1 CONTROL DE PETICIONES	GT10.1, RC2A-1, RC2A-2 RN2B y RN3B
SUBREQFPCI-09/2 REGISTROS DE ACCESO	GT10.2
<i>Continúa en siguiente página</i>	

<i>Continuación</i>	
Requerimiento	Equivalente PCI
SUBREQFPCI-09/3 AUTENTICACIÓN MULTIFACTOR	GT10.3
SUBREQFPCI-09/4 ACCESOS A NIVEL DE SISTEMA	GT10.4
SUBREQFPCI-09/5 ACCESOS ADMINISTRATIVOS	GT10.5
REQFPCI-10 MAPEOS DE TOKEN A TOKEN PROHIBIDOS	GT11
REQFPCI-11 PROTECCIÓN CONTRA VULNERABILIDADES COMUNES	GT12
REQFPCI-12 PRIMITIVAS CRIPTOGRÁFICAS USADAS	GT13
SUBREQFPCI-13/1 SOBRE EL CICLO DE VIDA	IT4A-1 y RC4B-1
SUBREQFPCI-13/2 DESCRIPCIÓN DEL PERÍODO CRIPTOGRÁFICO ACTIVO	IT4A-2 y RC4B-2
SUBREQFPCI-13/3 SOBRE LA DESTRUCCIÓN DE LAS LLAVES	IT4A-3 y RC4B-3
SUBREQFPCI-13/4 EXPORTAR LLAVE EN CLARO PROHIBIDO	RC1A-1
SUBREQFPCI-13/5 ENTROPÍA DE GENERACIÓN DE LLAVES	RC1A-2
SUBREQFPCI-13/6 LLAVES DE USO ÚNICO	RC1A-3
SUBREQFPCI-14/1 SOBRE EL MECANISMO GENERADOR (IRREVERSIBLES)	IT1A-1
SUBREQFPCI-14/2 CONTENIDO EN CLARO (IRREVERSIBLES)	IT1A-2
SUBREQFPCI-14/3 CREACIÓN DE UN DICCIONARIO (IRREVERSIBLES)	IT1A-3
SUBREQFPCI-14/4 SOBRE EL PROCESO DE AUTENTICACIÓN (IRREVERSIBLES)	IT1A-4
SUBREQFPCI-15/1 DISTRIBUCIÓN UNIFORME (CRYPTOGRÁFICOS)	RC1B-1
SUBREQFPCI-15/2 PERMUTACIÓN ALEATORIA (CRYPTOGRÁFICOS)	RC1B-2
SUBREQFPCI-15/3 CAMBIO DE LLAVE (CRYPTOGRÁFICOS)	RC1B-3
SUBREQFPCI-15/4 CAMBIO DE PAN (CRYPTOGRÁFICOS)	RC1B-4
<i>Continúa en siguiente página</i>	

<i>Continuación</i>	
Requerimiento	Equivalente PCI
SUBREQFPCI-15/4 CAMBIO DE PAN (CRIPTOGRÁFICOS)	RC1B-4
SUBREQFPCI-15/5 VERIFICACIÓN DE LA ALEATORIEDAD (CRIPTOGRÁFICOS)	RC1B-5
REQFPCI-16 ALMACENAMIENTO DE TOKENS (CRIPTOGRÁFICOS)	RC1C
REQFPCI-17 SEGURIDAD DE LA ADMINISTRACIÓN DE LLAVES (CRIPTOGRÁFICOS)	RC4A
REQFPCI-18 SOBRE LA LONGITUD DE LAS LLAVES (CRIPTOGRÁFICOS)	RC4C
REQFPCI-19 INDEPENDENCIA ESTADÍSTICA (CRIPTOGRÁFICOS)	RC4D
REQFPCI-20 GENERACIÓN Y ALMACENAMIENTO DE TOKENS (NO CRIPTOGRÁFICOS)	RN1A
SUBREQFPCI-21/1 DISTRIBUCIÓN EQUIPROBABLE (NO CRIPTOGRÁFICOS)	RN1B-1
SUBREQFPCI-21/2 PERMUTACIONES ALEATORIAS (NO CRIPTOGRÁFICOS)	RN1B-2
SUBREQFPCI-21/3 PARÁMETROS DE TOKENIZACIÓN (NO CRIPTOGRÁFICOS)	RN1B-3
SUBREQFPCI-21/4 VERIFICACIÓN DE LA ALEATORIEDAD (NO CRIPTOGRÁFICOS)	RN1B-5
REQFPCI-22 DISTRIBUCIÓN IMPARCIAL (NO CRIPTOGRÁFICOS)	RN1C
REQFPCI-23 INSTANCIAS ESTADÍSTICAMENTE INDEPENDIENTES (NO CRIPTOGRÁFICOS)	RN1D
REQFPCI-24 PROCESO DE DETOKENIZACIÓN (NO CRIPTOGRÁFICOS)	RN2A
REQFPCI-25 CIFRADO DE LA BASE DE DATOS (NO CRIPTOGRÁFICOS)	RN3A
REQFPCI-26 SEGURIDAD DE LA ADMINISTRACIÓN DE LLAVES (NO CRIPTOGRÁFICOS)	RN4A

Tabla 4.1: Resumen de requerimientos del PCI SSC para los sistemas tokenizadores.

4.1.4. Seguimiento de requerimientos de TOKENS

A continuación se encuentran los requerimientos que caen en la clasificación del IMPLEMENTACIONES mencionados en la sección 4.1. Se indica, para cada requerimiento, si el requerimiento fue satisfecho o no y cómo o por qué no.

- **REQPCI-06 DISTINCIÓN ENTRE TOKENS Y PERSONAL ACCOUNT NUMBER (PAN).** El programa tokenizador es capaz de distinguir entre un PAN y un TOKEN porque el segundo tiene el dígito de verificación calculado con el algoritmo de Luhn desfasado en uno (le suma uno al resultado). Por lo tanto, este requerimiento es satisfecho.
- **REQPCI-08 INTEGRIDAD DEL PROCESO DE TOKENIZACIÓN.** No hay mecanismos que garanticen la integridad del proceso de generación de TOKENS; por lo tanto, este requerimiento no es satisfecho.
- **REQPCI-10 MAPEOS DE TOKEN A TOKEN PROHIBIDOS.** El programa verifica que, cuando se va a realizar una tokenización, la entrada sea un PAN, por lo que no es posible pasar de un TOKEN a otro TOKEN sin haber detokenizado primero. Por lo tanto, este requerimiento es satisfecho.
- **REQPCI-11 PROTECCIÓN CONTRA VULNERABILIDADES COMUNES.** *Probablemente este requerimiento sea satisfecho por las banderitas del compilador, RQF7.*
- **REQPCI-12 PRIMITIVAS CRIPTOGRÁFICAS USADAS.** Los algoritmos que utiliza el programa tokenizador implementan estándares, por ejemplo, los cinco algoritmos utilizan AES y varios utilizan SHA-256. Por lo tanto, este algoritmo es satisfecho.
- **REQPCI-18 SOBRE LA LONGITUD DE LAS LLAVES (CRIPTOGRÁFICOS).** Las llaves utilizadas en los algoritmos son de, mínimo, 128 bits (en 3.2.4, se utiliza una de 256 bits); por lo tanto, este requerimiento es satisfecho.
- **REQPCI-25 CIFRADO DE LA BASE DE DATOS (NO CRIPTOGRÁFICOS).** Dado que este es un estudio propio de los algoritmos y su desempeño, y no de un sistema tokenizador, se decidió no cifrar los TOKENS en la base, pues ralentizaban mucho el desempeño. Por lo tanto, este requerimiento no es satisfecho.

4.2. Diseño de programa tokenizador

En la sección 3.2 se enlistaron y detallaron los algoritmos tokenizadores que implementaremos. En esta sección se describe, mediante UNIFIED MODELING LANGUAGE (UML), la estructura del programa generador de TOKENS.

En el diagrama de la figura 4.2 se muestra una vista estática general del programa. Las clases se dividen en dos paquetes: implementaciones y utilidades. Las utilidades abarcan estructuras de datos, interfaces,

clases de excepciones y clases de prueba; todas ellas no tienen que ver de forma directa con criptografía, sino que se trata de código de soporte para las implementaciones. En las implementaciones van no solamente los algoritmos presentados en la sección 3.2, sino que todas las demás clases (generalizaciones, utilidades, implementaciones de soporte) que están relacionadas con los algoritmos tokenizadores; en el caso de este paquete, sí todo está relacionado o con criptografía o con un entorno bancario.

La figura 4.1 muestra la dependencia entre los paquetes que componen al programa: las implementaciones importan el contenido de las utilidades; ambos paquetes cuentan con equivalentes de prueba (más adelante se detalla el funcionamiento de las pruebas), en donde cada uno importa el paquete que está probando.

Retornando al diagrama 4.2: muestra solamente las clases e interfaces directamente relacionadas con los algoritmos tokenizadores; en las próximas secciones se mostrarán algunas otras vistas. Las utilidades contienen las interfaces para definir los distintos tipos de funciones y la estructura de datos (Arreglo) que se usa de manera constante en todo el programa. Las implementaciones muestran las interfaces que definen y clasifican a los algoritmos tokenizadores; también se muestra (por razones de espacio, solo el nombre) las clases concretas de los sistemas tokenizadores.

Las interfaces de las funciones utilizan plantillas para definir las propias firmas de los métodos abstractos que declaran. Esta es una forma bastante efectiva para hacer diseños débilmente acoplados (ver ACOPLAMIENTO): por ejemplo, las redes Fesitel necesitan de una función de ronda sobre la que *no necesitan* saber nada en particular, solo necesitan saber que la clase define un método **operar**, que es con lo que funciona el algoritmo definido por la red.

El Arreglo es una implementación propia de una estructura de datos de almacenamiento secuencial. Es solamente una interfaz a una sección de memoria (un arreglo tradicional); sin embargo, lo importante de este es que imita el formato de las estructuras de la librería estándar de C++ (esquema de constructores y destructores para gestión de memoria, uso de plantillas para programación genérica, etcétera). El arreglo de dígitos es una especialización que mantiene una representación interna tanto en cadena como en número; este arreglo es el que ocupan para comunicarse todos los algoritmos tokenizadores. Por esta última razón es por la que se mantienen las dos representaciones internas: algunos métodos requieren interpretar las entradas como números y otros como cadenas.

Los algoritmos tokenizadores implementan la interfaz definida por las funciones con inverso definiendo el tipo de ida y de vuelta como un arreglo de dígitos (PERSONAL ACCOUNT NUMBER (PAN) y TOKEN). Las interfaces intermedias (algoritmos reversibles e irreversibles) permiten definir un comportamiento genérico para cada tipo de algoritmo; estas a su vez declaran los métodos abstractos que los algoritmos concretos deben de implementar.

El diagrama de la figura 4.3 muestra la división en componentes del programa. El componente principal

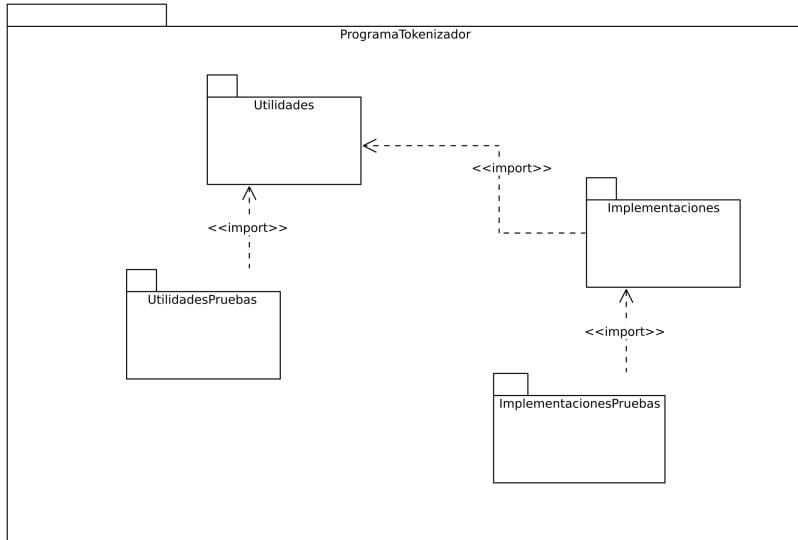


Figura 4.1: Diagrama de dependencias entre paquetes.

es el llamado *Programa tokenizador*; este se comunica con el exterior mediante la interfaz de un algoritmo tokenizador (el puerto del costado derecho), y depende de un componente que implemente la interfaz de CARD DATA VAULT (CDV) (el puerto del costado inferior). Existen varios clientes del componente principal: un ejecutable para generar tokens, el programa de pruebas y el programa de comparación de desempeño. En el diagrama también se muestra la composición interna del componente principal: la interfaz pública se divide en otras dos interfaces: la de los algoritmos reversibles y la de los irreversibles. En las próximas secciones se muestran los detalles de cada uno de los componentes internos.

4.2.1. Clases de FFX

En la figura 4.4 se muestran las clases que conforman al módulo de FFX. Sin contar a las tres interfaces de funciones (que forman parte del paquete de utilidades) todas las clases son del paquete de implementaciones.

La clase de la red Feistel implementa la interfaz de una función con inverso (tiene tanto operación de ida, como de vuelta) y se compone (por medio de una relación de composición) de una función, utilizada como función de ronda, y de una función con inverso, utilizada como operador de combinación. Ambas clases hijas (redes Feistel alternantes o desbalanceadas) contienen un indicador de desbalanceo y sobreescreiben ambas operaciones de la superclase; la red feistel alternante (ver sección 2.2.3) agrega una nueva función de ronda: una para las pares y otra para las impares.

En la parte superior de diagrama se muestran las dos posibles operaciones de combinación que soporta ffx: una a nivel de bloque y la otra nivel de carácter. La única clase en donde estas dos opciones son restringidas es desde la construcción de FFXA10; el desacoplamiento dado por las interfaces permite que lo único que necesita saber la red es que tiene una función que recibe un arreglo y entrega un arreglo.

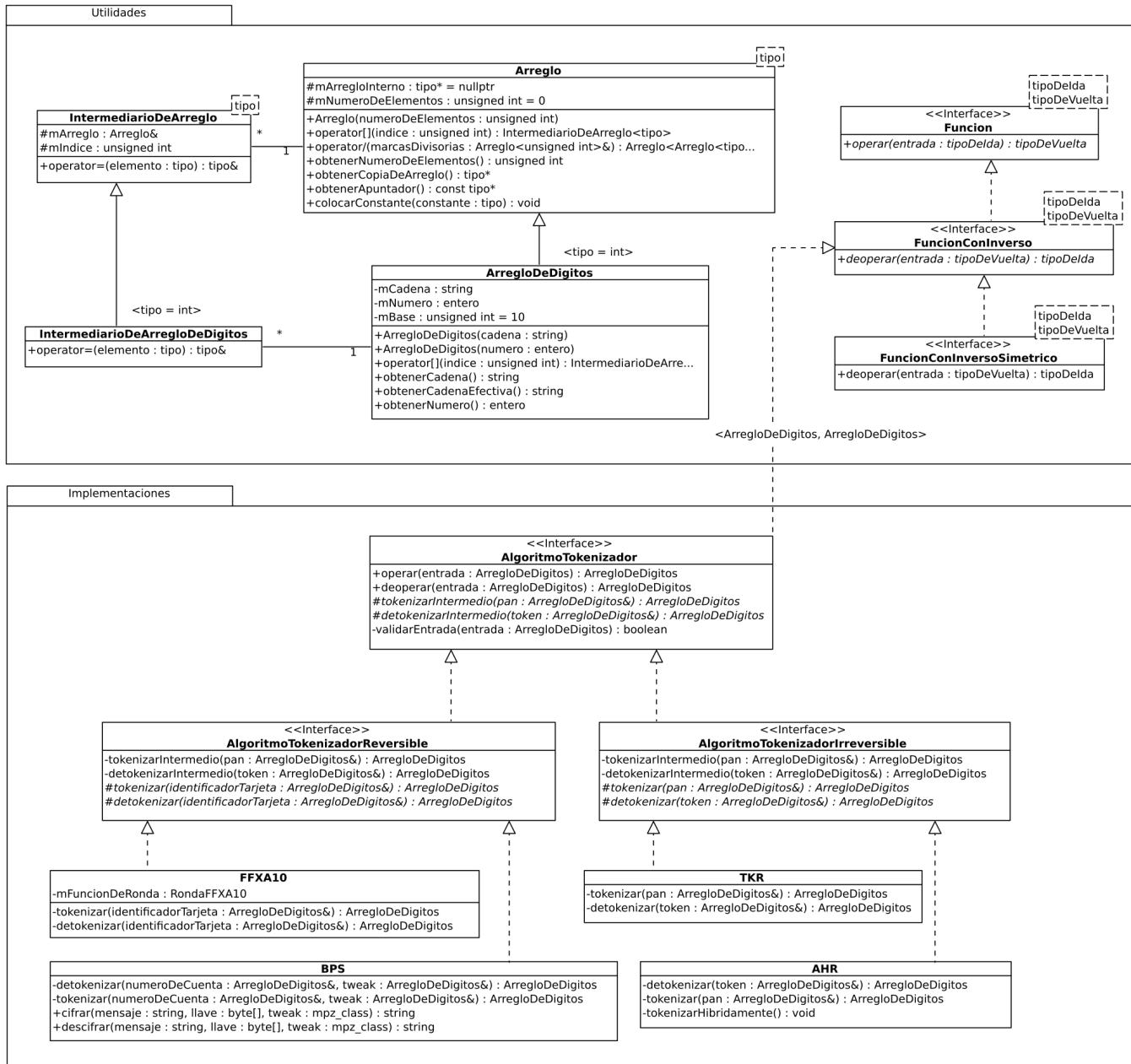


Figura 4.2: Diagrama de clases general.

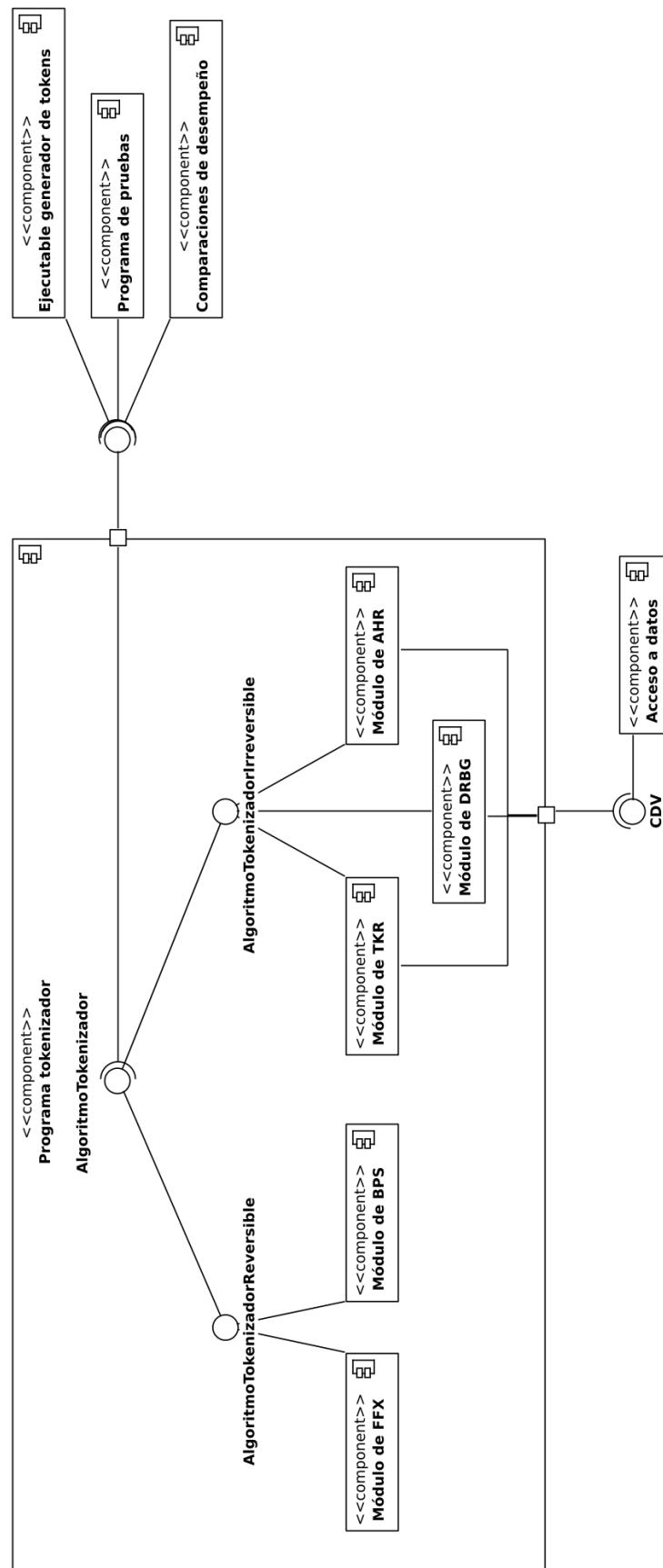


Figura 4.3: Diagrama de componentes de programa.

Otra clase con la misma estructura que las dos anteriores es la de la función de ronda de FFXA10: implementa la interfaz de una función con inverso simétrico (que a su vez implementa un función con inverso); una instancia de esta clase en FFX es usada para construir a la red Feistel con la que se opera.

La clase de FFX es solamente un medio de comunicación con la red Feistel interna, esto es, no agrega ninguna lógica extra a los procesos de operación y operación inversa. Lo mismo ocurre con FFXA10; solo que mientras el constructor de la superclase es abierto a personalizar la red (FFX debería servir para cualquier cifrado que preserva el formato, no solo para un alfabeto de dígitos), el constructor de FFXA10 tiene parámetros bastante limitados, y su construcción de la superclase y la red Feistel ya es predefinida según la descripción de esta colección hecha en la sección 3.2.1. Es esta última clase (FFXA10) la que implementa la interfaz de los algoritmos tokenizadores reversibles.

4.2.2. Clases de BPS

Primero, se tiene que mencionar que a pesar de que BRIER-PEYRIN-STERN (BPS) puede verse como una versión más específica de FORMAT-PRESERVING FEISTEL-BASED ENCRYPTION (FFX), esta clase no se realizó de forma directa a partir de la clase de la red Feistel.

A BPS se le puede considerar como un modo de operación que usa un cifrado interno BC, que a su vez usa un cifrado de ronda, situación por la que BPS es una clase que implementa el modo de operación descrito en 3.2.2, que utiliza la clase BC, cuyo funcionamiento se describe en 3.2.2, el cual se puede resumir como una red Feistel que opera sobre cadenas construidas a partir de cualquier alfabeto, en vez de sobre cadenas binarias.

En cuanto al cifrado de ronda, esta clase se construyó usando la librería de Cripto++, dando la posibilidad de usar ADVANCED ENCRYPTION STANDARD (AES) o DATA ENCRYPTION STANDARD (DES).

El diagrama de clases del módulo de BPS, sin contemplar las clases del paquete de utilidades, se muestra en la figura 4.5.

4.2.3. Clases de TKR

En la figura 4.6 se muestran las clases relacionadas al módulo de TKR. Al igual que con el diagrama anterior, la gran mayoría de las clases pertenecen al paquete de implementaciones; las excepciones marcan el paquete al que pertenecen.

Este es el primer diagrama en donde es necesario mostrar el esquema de acceso a una fuente de datos. Este es descrito por una interfaz llamada CDV (CARD DATA VAULT), que define las operaciones que debe de proveer cualquier fuente de datos que se desee usar dentro del programa. En un primer esquema existen dos clases concretas que implementan esta interfaz: un acceso trivial y un acceso a MySQL (ver sección 5.1). El

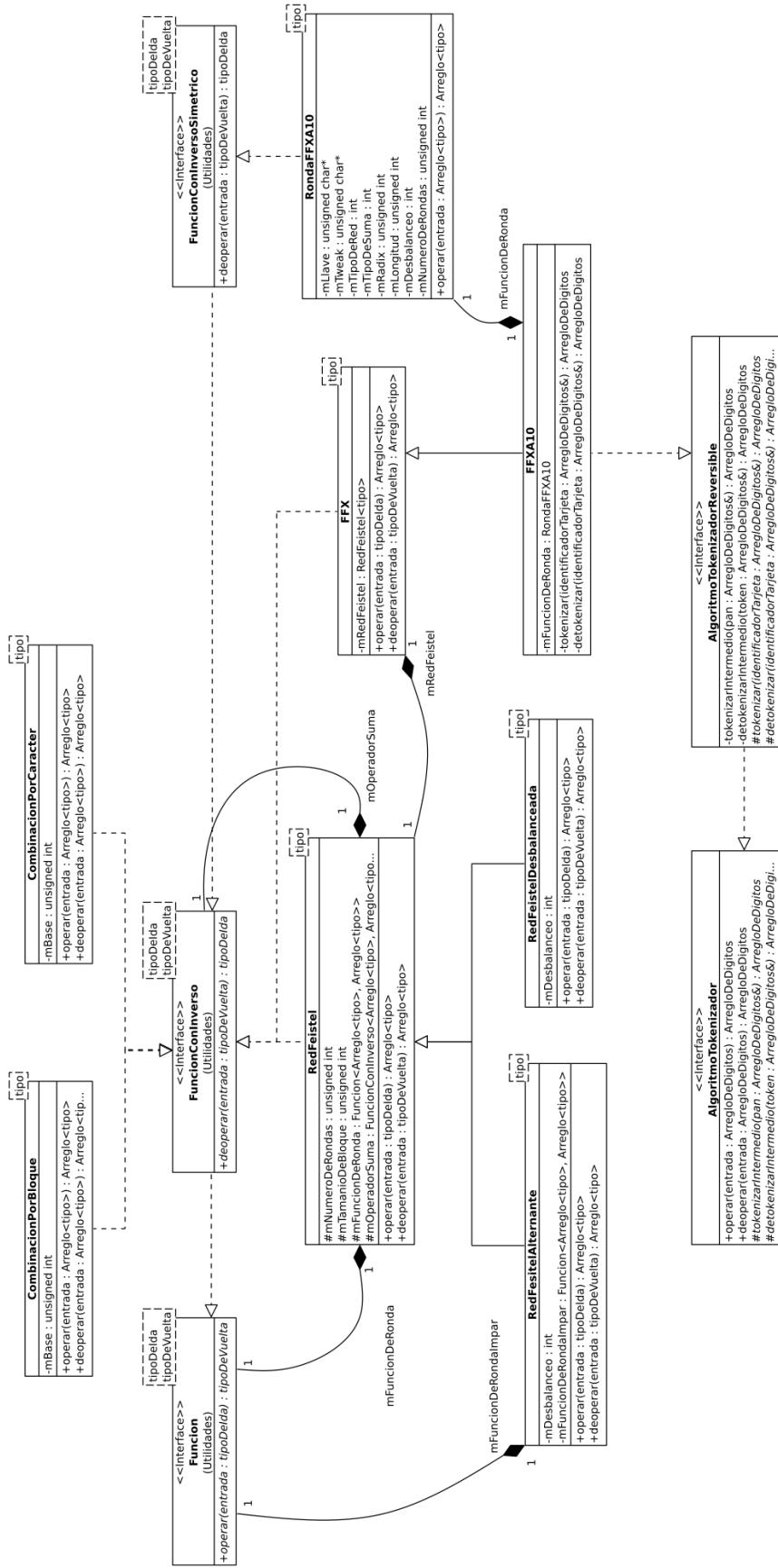


Figura 4.4: Diagrama de clases de módulo de FFX.

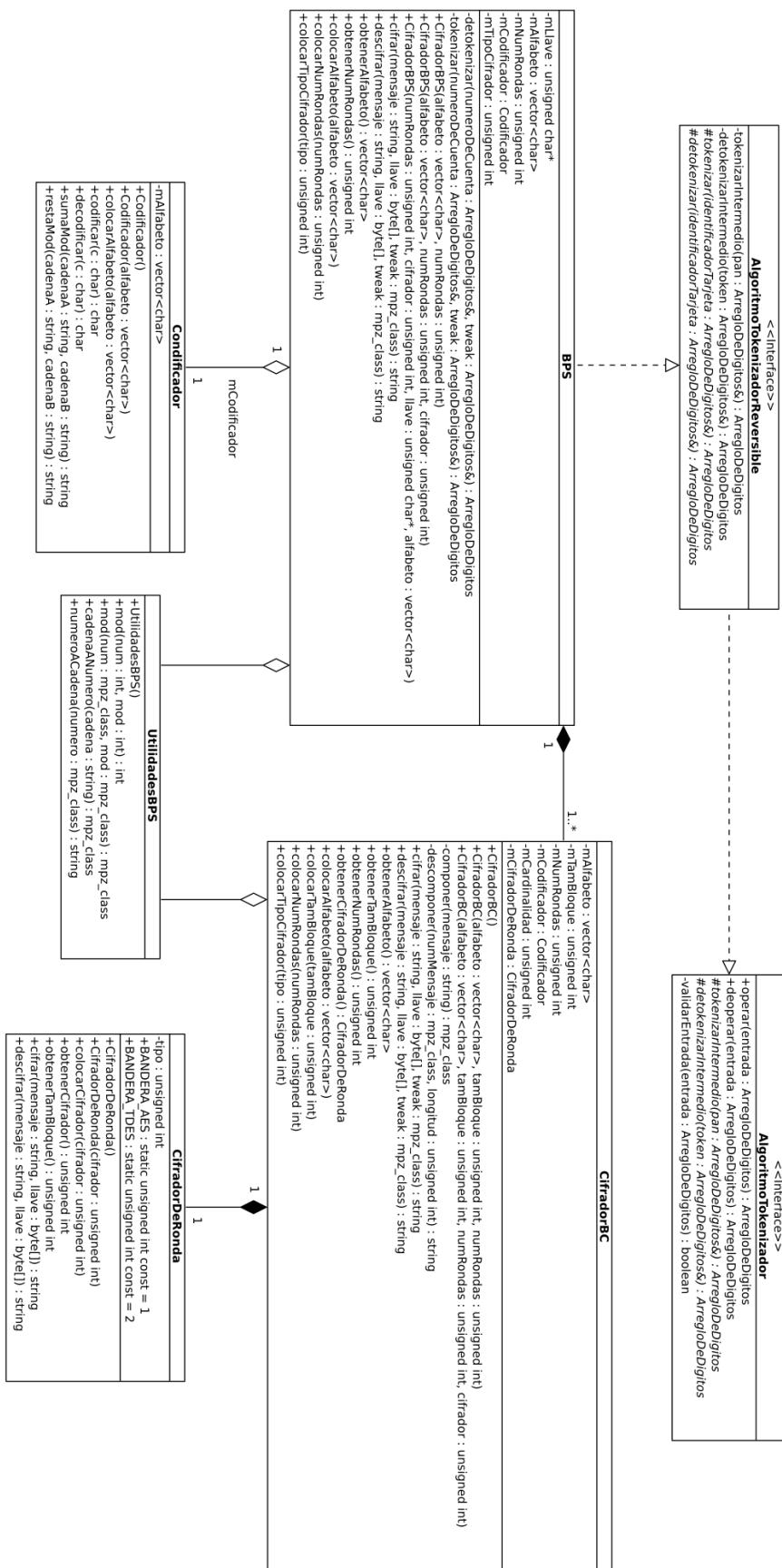


Figura 4.5: Diagrama de clases de módulo de BPS.

acceso trivial permite definir métodos «de prueba» cuando se introduce una nueva operación en la interfaz; de esta forma, el código cliente puede ocupar esta implementación en lo que se desarrolla la verdadera, con conexión a base de datos. Esta manera de separar interfaz de implementación para el acceso a datos es un patrón de arquitectura conocido como DATA ACCESS OBJECT (DAO); a grandes rasgos permite separar la definición de *qué* datos necesita la aplicación, del *cómo* los obtendrá; de esta manera, un cambio en el gestor de base de datos no debe afectar al código que usa la interfaz.

En la parte inferior del diagrama se muestra la clase *Registro*. Esta representa la única clase de datos del entorno del modelo; contiene la relación entre un PERSONAL ACCOUNT NUMBER (PAN) y un TOKEN. Como aclaración, es la única clase de datos del modelo del módulo del programa tokenizador; la interfaz web introducirá muchas otras clases al modelo.

TKR, como se explicó en la sección 3.2.3, necesita de una función que genere tokens pseudoaleatorios (la función RN) la cuál necesita de una función proveedora de bits pseudoaleatorios. En el diagrama se muestran tres versiones de esta última función: una trivial, una basada en ADVANCED ENCRYPTION STANDARD (AES) y la tercera basada en un DETERMINISTIC RANDOM BIT GENERATOR (DRBG) (costado izquierdo del diagrama). La trivial es una primera implementación que genera números pseudoaleatorios de un modo que no es criptográficamente seguro. La basada en AES es tal cuál se describe en el artículo [45]. Por último, la basada en un DRBG, es la que permite definir el algoritmo tokenizador que se describe en la sección 3.2.5; para este algoritmo se ocupa la misma estructura que para TKR, sin embargo, lo que importa en esta implementación es la propia creación del generador, que se muestra en secciones posteriores.

La relación entre la función RN y su mecanismo de generación interno está totalmente desacoplada: lo único que necesita la clase de la función RN es otra función que reciba enteros sin signo y entregue arreglos de bytes. Este modelo de desacoplamiento no se imitó en la relación entre la clase de TKR y la función RN porque la especificación de TKR es muy específica respecto a qué función usar.

La clase de TKR implementa la interfaz de un algoritmo tokenizador irreversible, por lo que debe (programación por contrato) implementar los métodos para tokenizar y detokenizar.

4.2.4. Clases de AHR

AHR utiliza la interfaz CDV para poder interactuar con la base de datos y realizar el proceso de detokenización, pues es un algoritmo irreversible. Hace uso también de algunas utilidades como potencias y el cálculo del dígito verificador mediante el algoritmo de Luhn desfasado. Utiliza también la clase del registro al momento de agregar un nuevo token a la base de datos y la clase de *Arreglo de dígitos* para implementar los métodos de *tokenizar* y *detokenizar* definidos por la interfaz *Algoritmo tokenizador irreversible*.

Finalmente, se utiliza la clase AES para realizar el cifrado por bloque; esta clase es una interfaz y permite utilizar el cifrado de AES por hardware si el procesador tiene las instrucciones necesarias. En caso

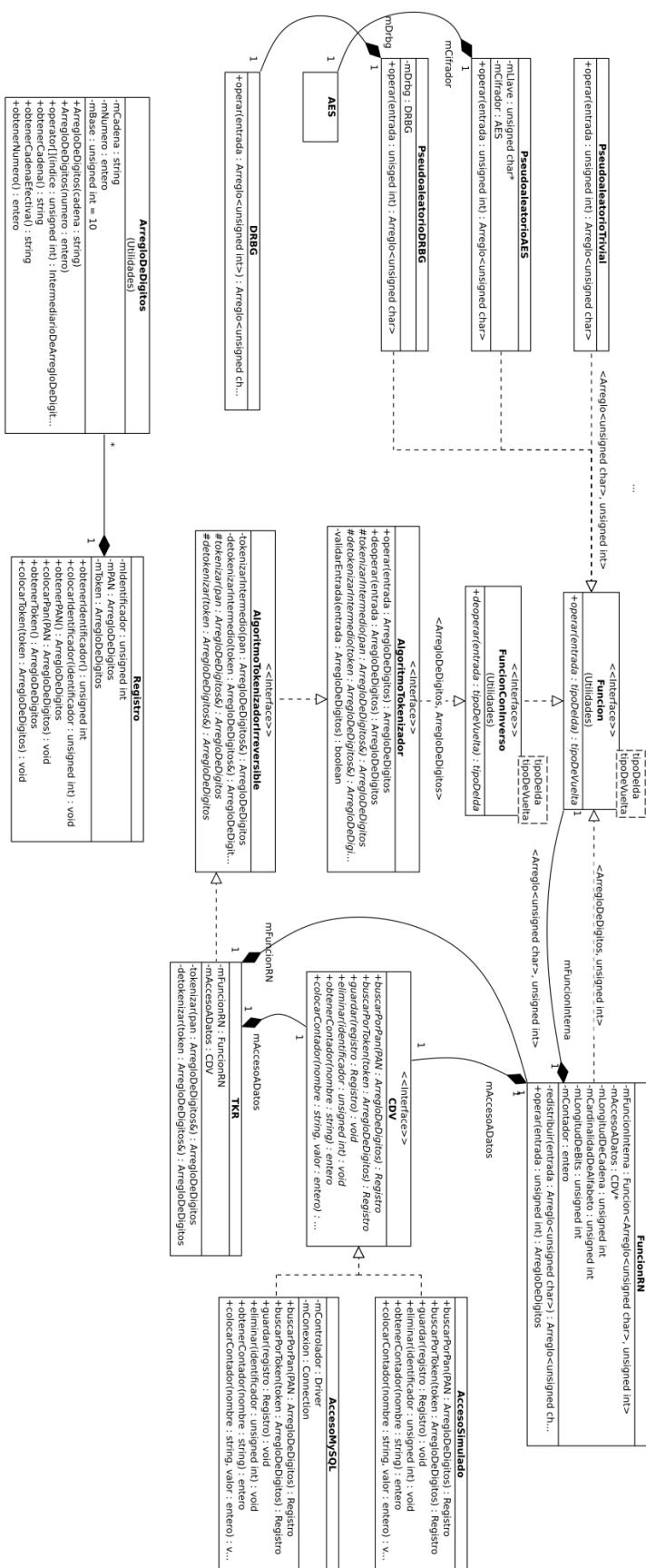


Figura 4.6: Diagrama de clases de módulo de TKR.

de no tenerlo, cifra mediante las funciones de CryptoPP.

El diagrama de clases de AHR se encuentra en la figura 4.7.

4.2.5. Clases de DRBG

Las clases relacionadas con el generador de números pseudoaleatorio se muestran en la figura 4.8. Nuevamente, todas las clases pertenecen al paquete de las implementaciones.

Un DETERMINISTIC RANDOM BIT GENERATOR (DRBG) implementa la interfaz de una función que recibe enteros sin signo y entrega arreglos de bytes. Esta clase define la estructura general de un generador: las cinco funciones definidas por el estándar del NIST (ver sección B) y el conjunto de valores miembro que conforman el estado del generador. El DRBG declara la función abstracta para generar bytes; todos los posibles generadores concretos deben implementar esta función para poder tener un generador completo.

La clase del generador necesita una fuente de entropía (o fuente de aleatoriedad). Siguiendo el esquema de débil acomplamiento dado por las interfaces de las funciones, la fuente de entropía es una función que recibe un entero y regresa un arreglo de bytes. En el costado izquierdo del diagrama se muestran dos de las posibles fuentes de entropía: aleatoriedad trivial y aleatoriedad por hardware (para más detalles, ver sección 5.1.3).

Existen dos implementaciones concretas para el generador de números pseudoaleatorios: una basada en una función hash y la otra basada en un cifrado por bloques (sección 3.2.5). Ambos agregan sus propios datos miembro al estado del generador, sobrecargan las funciones de cambio de semilla y desintanciación e implementan el método abstracto de la superclase. También se utilizan enumeraciones para indicar al código cliente cuales son las distintas funciones hash y cifrados por bloque que pueden ser ocupados.

4.2.6. Estructura de pruebas de unidades

A la par que se implementa el programa descrito hasta ahora, se crearán PRUEBAS DE UNIDADES y PRUEBAS DE COMPONENTES. La idea de combinar el desarrollo de pruebas con el proceso de implementación es encontrar problemas en una etapa temprana, facilitar el cambio de código existente y simplificar la integración de nuevos componentes. La ejecución de las pruebas está ligada a un proceso de integración continua, de forma que con cada pequeño cambio (con cada *commit*) se ejecute desde cero todo el proceso de compilación y pruebas.

Algunos lenguajes (como Python) cuentan con un modelo de pruebas integrado; otros (como Java o Javascript) cuentan con librerías muy populares (JUnit o Karma). En el caso de C++ no hay ningún soporte desde el propio lenguaje, y entre las librerías existentes tampoco hay ninguna que se haya establecido como favorita. Es por eso que se decidió crear una estructura de pruebas propia: en la figura 4.9 se muestra la

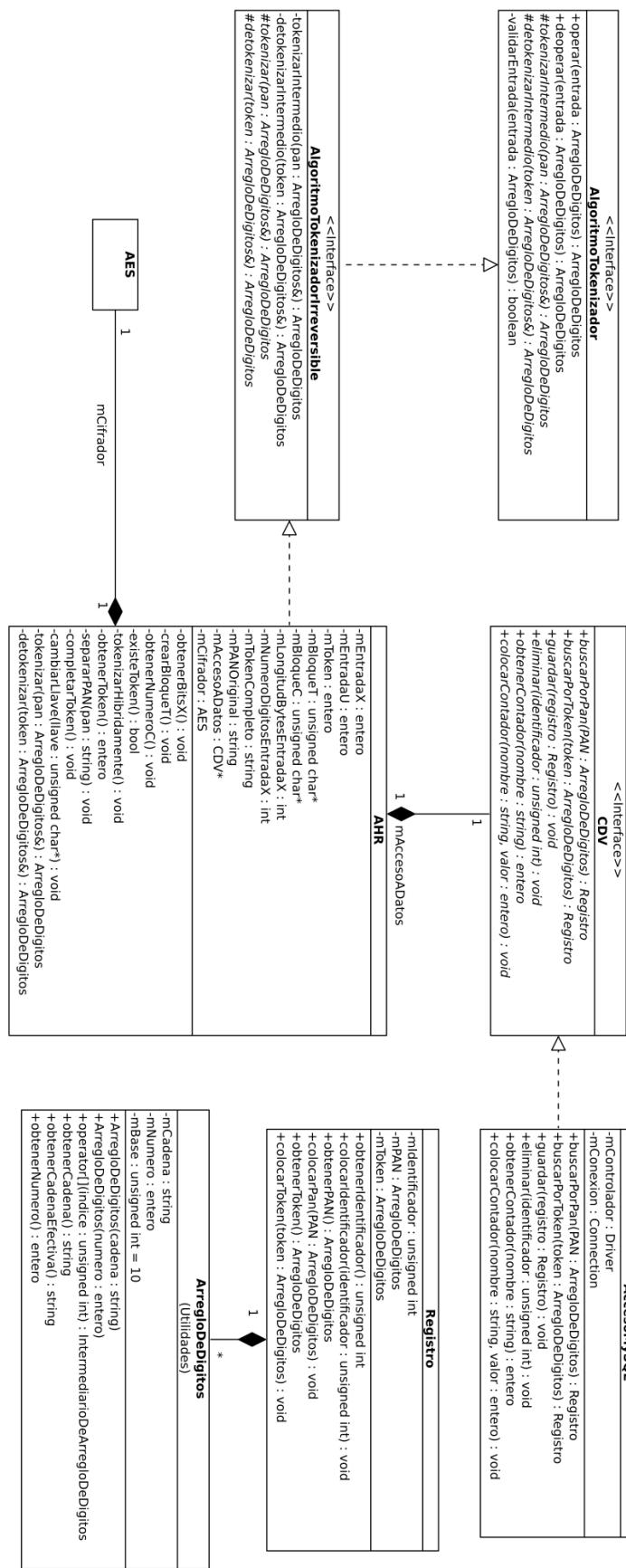


Figura 4.7: Diagrama de clases de módulo de AHR.

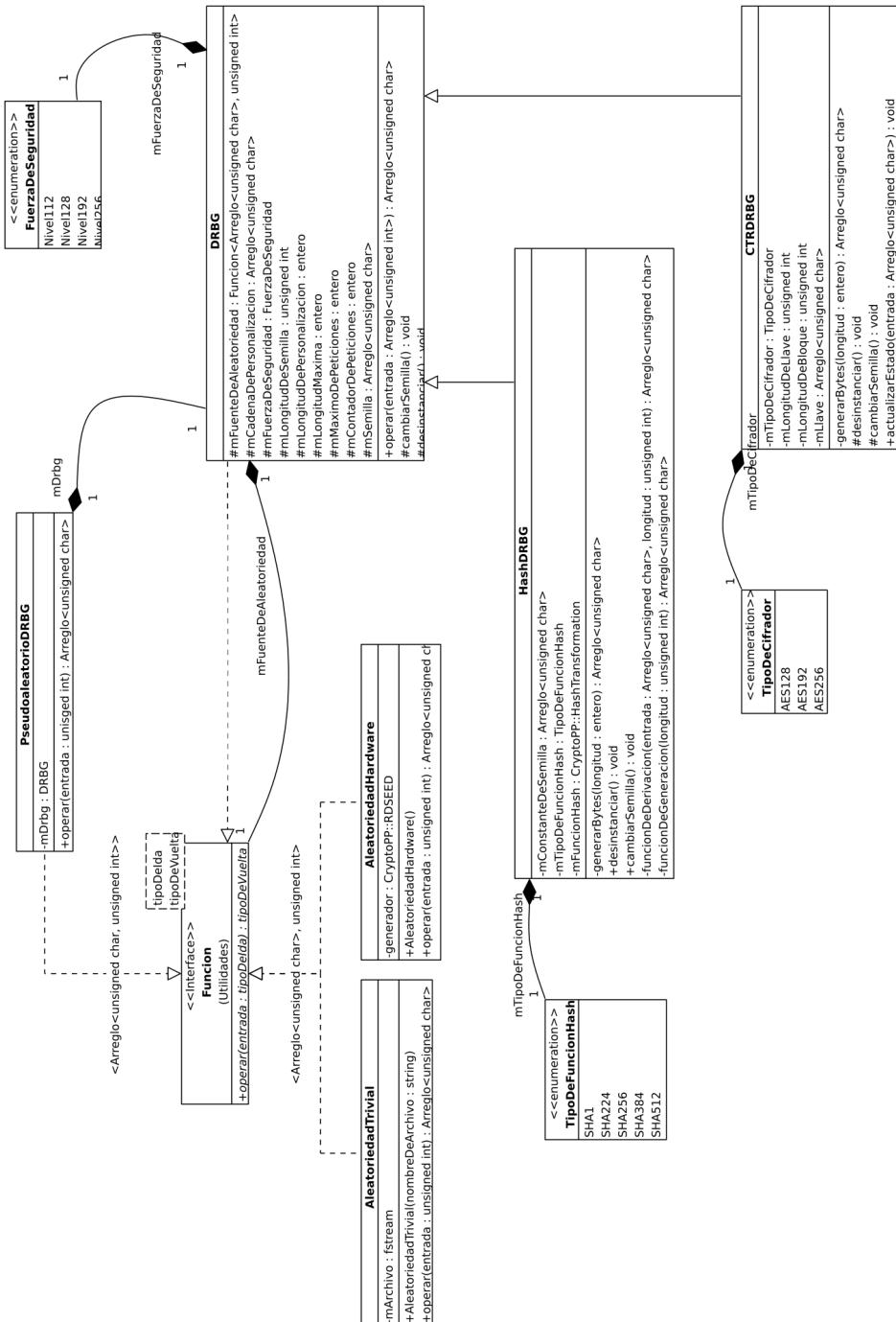


Figura 4.8: Diagrama de clases de módulo de DRBG.

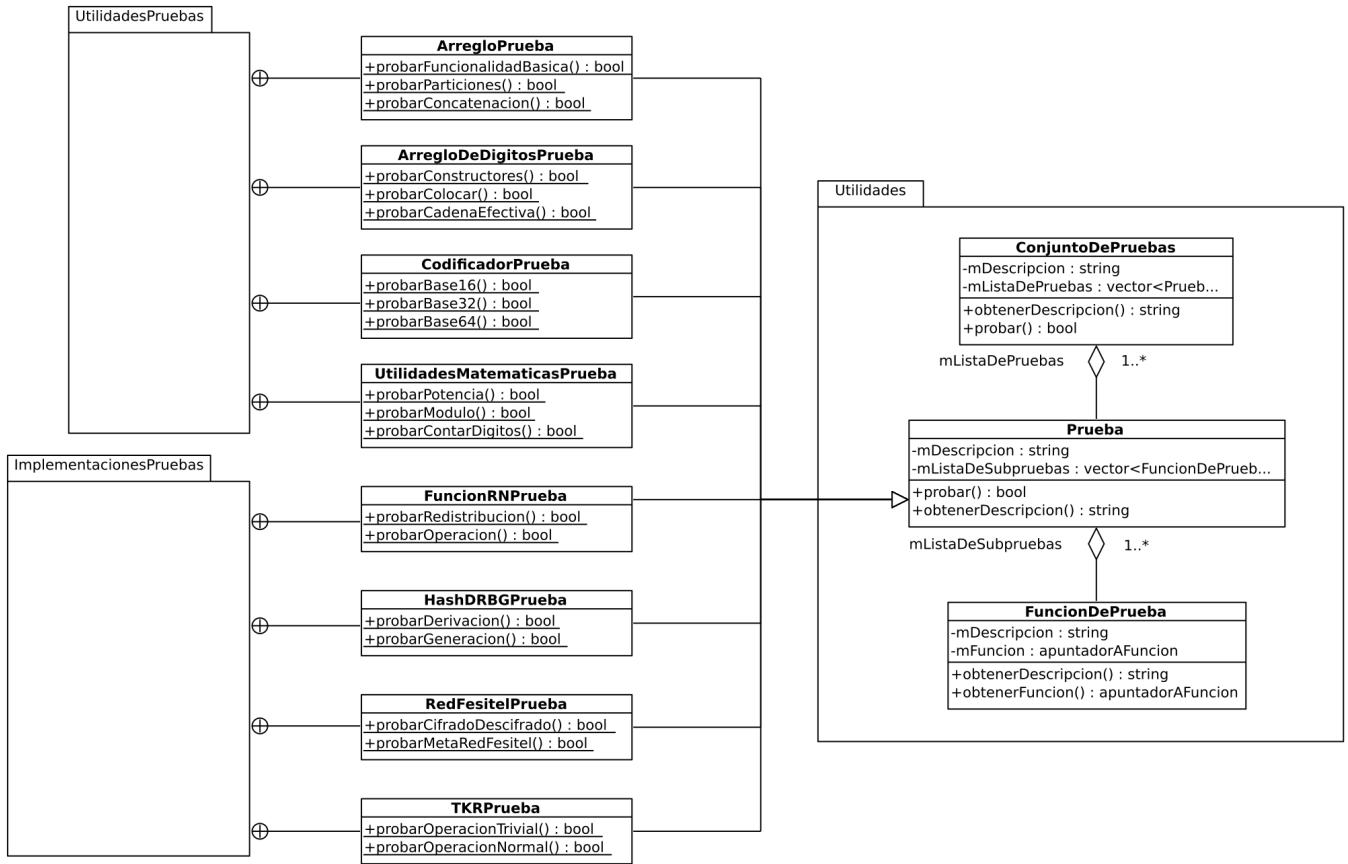


Figura 4.9: Diagrama de clases de la estructura de pruebas.

estructura de las clases de pruebas para todo el programa. Este muestra a modo de ejemplo solamente algunas de las clases; todas las demás siguen el mismo esquema.

Las tres clases de la derecha, dentro del paquete de utilidades, son para agrupar y gestionar de una forma genérica a todas las pruebas. Un conjunto de pruebas tiene un lista de la clase Prueba. La clase Prueba tiene una lista de funciones de prueba. Las funciones de prueba tienen una descripción y un apuntador a una función (la propia prueba). Todas las pruebas dentro del programa deben ser especificaciones de la clase prueba. El estándar es que una clase de prueba defina todas las funciones (booleanas y estáticas) que necesite y las agregue a la lista de pruebas de la superclase.

Existen dos paquetes para agrupar a las pruebas: uno para lo que hay en implementaciones y otro para las utilidades. Lo ideal es que por cada parte del código en donde haya lógica no trivial se hagan funciones de prueba. Así, si hay una clase *Ejemplo*, debe de existir una clase *EjemploPrueba* que capture el comportamiento esperado de *Ejemplo*. En algunos casos, las clases de prueba deben tener acceso a lo que la propia clase encapsula (miembros privados o protegidos), por lo que la clase de prueba debe ser amiga de la clase probada.

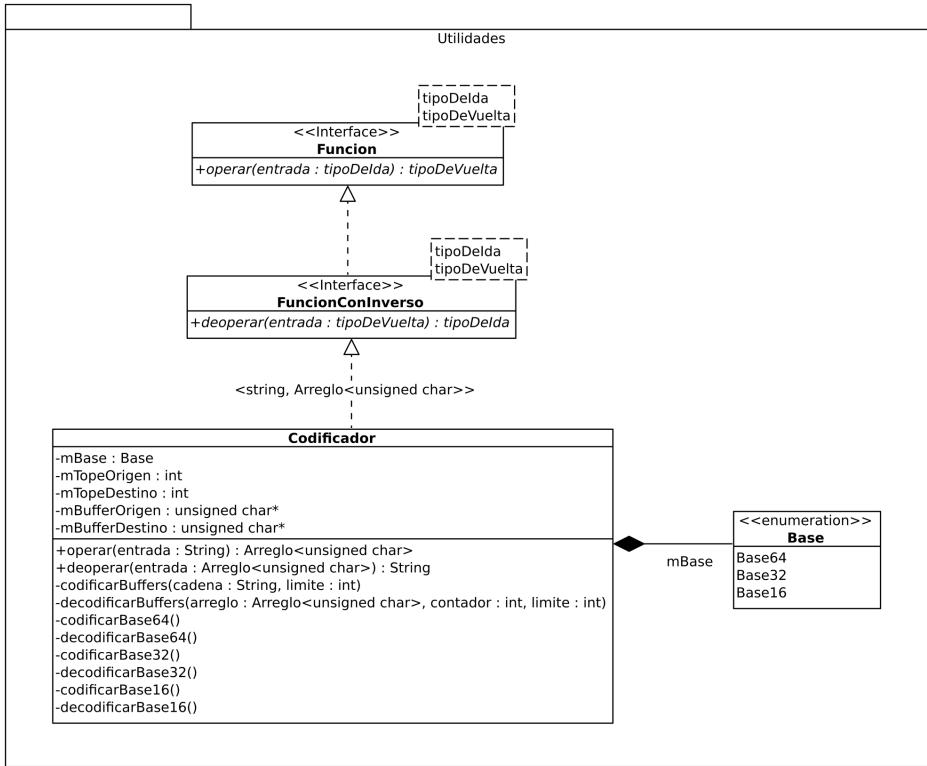


Figura 4.10: Clase de codificador.

4.2.7. Codificador de binario a ASCII

Para evitar almacenar en formato binario las llaves generadas por el ejecutable tokenizador, se implementó el codificador de binario a caracteres ASCII descrito en el REQUEST FOR COMMENTS (RFC) 4648 [49]. La idea es no guardar una llave en su representación binaria, sino guardar solamente caracteres imprimibles. Para esto, el codificador hace un mapeo de los bytes originales (rango de 256 números) a bytes acortados, que se encuentren en un rango de 64, 32 o 16 números.

En la figura 4.10 se muestran la clase del codificador junto con su relación con otras clases. Existe una enumeración para indicar las distintas posibilidades de codificación. El codificador implementa a una función con inverso, la cuál a su vez implementa a una función.

Capítulo 5

Implementación de programa tokenizador

«There are two methods in software design. One is to make the program so simple, there are obviously no errors. The other is to make it so complicated, there are no obvious errors.»

TONY HOARE.

En este capítulo se presentan algunos de los aspectos más interesantes de la implementación del programa descrito en el capítulo 4. Comienza describiendo las tecnologías usadas para la implementación y desarrollo del proyecto. También se presentan los resultados de las comparaciones de desempeño y de las pruebas estadísticas de los generadores pseudoaleatorios.

5.1. Tecnologías

Las primeras decisiones tomadas con respecto a la implementación y al diseño fueron sobre el paradigma de programación y el lenguaje a utilizar. Para tomar estas decisiones se tomaron en cuenta varias consideraciones: primero, el paradigma orientado a objetos ofrece considerables ventajas con respecto a la programación estructurada, por lo que, dentro de lo posible, se buscaría un lenguaje con soporte a este paradigma; segundo, las implementaciones criptográficas tienen altos requerimientos de rendimiento, por lo que, de los posibles lenguajes, se necesitaba elegir uno que, aunque no el más rápido, sí se encontrara entre los de mayor velocidad.

A lo largo de la carrera se ha tenido contacto con bastantes lenguajes que, si bien no se dominan, sí se posee una base firme como para ser usada: C, C++, Java, Python, Javascript y PHP. Por la cuestión del rendimiento, todos los lenguajes interpretados (Python, Javascript, PHP) quedaron fuera de consideración; la decisión (dentro de los lenguajes con soporte al paradigma orientado a objetos) quedó entre Java y C++: Java (aún en las últimas versiones, en donde se han hecho considerables progresos) es mucho más lento que C++, dado que el trabajo de la máquina virtual en tiempo de ejecución es bastante considerable; por lo tanto, si de un lenguaje orientado a objetos se trataba, sería C++. La última decisión se dió entre C y C++: orientación a objetos contra rendimiento. Al final se optó por la orientación a objetos: aún cuando C es más rápido que C++, la diferencia no es tan grande, mientras que un programa con un buen diseño orientado a objetos sí puede ser mucho más mantenable que uno con un enfoque estructurado.

5.1.1. Dependencias de distribución

A continuación se enlistan las principales dependencias del proyecto. Para cada una se da una breve argumentación sobre su uso, se especifica, en donde aplica, la licencia que tiene, y se coloca la versión ocupada.

DEPPT-01 Compilador: GCC v7.3.

Licencia: GPL v3

Página oficial: [HTTPS://GCC.GNU.ORG/](https://gcc.gnu.org/)

La razón principal de su uso es porque se trata del compilador por defecto de los sistemas operativos GNU/Linux. La versión de C++ que se utiliza es C++14 (opción `-std=c++14` del compilador). Por razones de compatibilidad con algunas otras dependencias (en particular, el conector de la base de datos) no es posible utilizar la última versión al momento, C++17.

DEPPT-02 Gestor de base de datos: MariaDB v10.1.

Licencia: GPL v2

Página oficial: <https://mariadb.org/>

El gestor de base de datos que más se ha utilizado en la carrera es MySQL, por lo que es el seleccionado para el programa tokenizador; en realidad, se trata de una bifurcación de MySQL: MariaDB, cuyo uso ha sido ampliamente difundido en varias distribuciones de GNU/Linux.

DEPPT-03 Librería criptográfica: Crypto++ v7.

Licencia: BSL v1

Página oficial: <https://cryptopp.com/>

Otra decisión importante a tomar antes de hacer las implementaciones de los algoritmos tokenizadores fue la librería de funciones criptográficas que se utilizaría. De todas las posibles librerías que están validadas por el NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY (NIST) y que cuentan con una licencia para el uso público, al final se hicieron pruebas con dos: Openssl y Crypto++. Openssl está escrita en C y cuenta con un historial que le da mucha reputación (es la librería que utiliza Openpgp y Openssh). Crypto++ cuenta con una extensa red de colaboradores, implementa una gran cantidad de algoritmos y está escrita sólo en C++. De ambas, la que se encontró más fácil de usar fue Crypto++, por lo que es la que se utiliza en las implementaciones.

DEPPT-04 Librería alrededor de AES-NI: libaesni.

Licencia: Sin licencia

Página oficial: <https://github.com/amiralis/libaesni>

Envoltura en C alrededor de las instrucciones de ensamblador de Intel para ADVANCED ENCRYPTION STANDARD (AES) (sección 5.1.3). La razón de ser de esta librería dentro de las dependencias es para evitar tener que escribir código directamente en ensamblador. Este programa funciona como interfaz en C con las instrucciones de Intel.

5.1.2. Dependencias de desarrollo

A continuación se enlistan las dependencias de desarrollo. Aunque estas no son necesarias para el producto final, sí lo son para el cambio y mantenimiento del programa. Muchas de estas dependencias se usan como servicios (v. gr. github o travis). Las elecciones de estas dependencias no siguieron siempre un proceso de eliminación como las anteriores, sino que fueron seleccionadas por cuestión de facilidad de uso (de los autores).

DEPPT-05 Control de versiones: Git v2.17.

Licencia: GPL v2

Página oficial: <https://git-scm.com/>

Es el VERSION CONTROL SYSTEM (VCS) más común en entornos GNU/Linux. Es muy importante

su integración con Github.

DEPPT-06 Servicio de *hosting* para el VCSVCS: Github.

Licencia: No aplica (servicio)

Página oficial: <https://github.com>

Aunque a la fecha soporta otros tipos de VCS, originalmente era exclusivo a Git. Su función principal es mantener una copia remota del código, desde la cual un desarrollador baja la última versión y en donde actualiza sus cambios. También se llegaron a utilizar algunas funcionalidades adicionales, como el reporte de problemas, los comentarios en código y la integración con Travis.

DEPPT-07 Herramienta de integración continua: Travis.

Licencia: No aplica (servicio)

Página oficial: <https://travis-ci.org/>

Entre sus principales tareas están compilar desde cero el proyecto, ejecutar todas las pruebas, generar y desplegar la documentación y el reporte de vuelta a github, ejecutar las pruebas de desempeño y guardar los resultados en la base de datos.

DEPPT-08 Documentación de código: Doxygen.

Licencia: GPL v2

Página oficial: <https://www.stack.nl/~dimitri/doxygen/>

Herramienta para generar documentación del código. Permite mantener la documentación en los mismos archivos fuente, ya que la salida es generada a partir de comentarios. Se cuentan con dos versiones, una en HTML (https://rjf7.github.io/proyecto_lovelace/documentacion_doxxygen/html/index.html) y la otra como PDF (https://rjf7.github.io/proyecto_lovelace/documentacion_doxxygen/latex/refman.pdf).

5.1.3. Instrucciones de ensamblador

Las implementaciones de los algoritmos tokenizadores dependen de un par de conjuntos de instrucciones desarrollados por Intel: el primero es AES-NI y el segundo RD-SEED.

AES-NI son siete instrucciones que implementan directamente, a nivel de hardware, algunos de los procesos del algoritmo de AES (sección 2.2.5). Estas instrucciones fueron sacadas al mercado en septiembre de 2010, para la familia de procesador de microarquitectura de 32nm[50]. Su objetivo es acelerar la operación de AES, una de los cifrados por bloque más utilizados.

RD-SEED se publicó junto con otra instrucción, RD-RAND. RD-SEED es un generador de números aleatorios que obtiene entropía de procesos físicos relacionados con el procesador; cumple con los estándares definidos por el NIST en [34] (sección B). Su función es servir como fuente de entropía a generadores pseudoaleatorios. RD-RAND es un generador de números pseudoaleatorios que cumple con el estándar del

NIST 800-90A. Ambas instrucciones estuvieron disponibles a partir de 2014, con la versión de procesadores de microarquitectura *broadwell* [50]. Para los fines de este proyecto, la instrucción que ocupamos es RD-SEED; RD-RAND es equivalente a nuestra implementación del generador pseudoaleatorio (sección 5.2.5). También es importante recalcar que RD-RAND, por su naturaleza de algoritmo, se encuentra disponible en casi todos los procesadores de Intel y AMD, mientras que RD-SEED implica funcionalidades en hardware que solo soportan algunas versiones de procesadores Intel.

Para ambas funcionalidades, el enfoque de implementación busca ser lo más flexible posible: si la arquitectura en la corre el programa soporta las instrucciones en cuestión, perfecto, sino, se buscan alternativas que de todas formas permitan ejecutar el programa. El caso de AES-NI, la alternativa es utilizar la implementación de AES de Cryptopp. Para RD-SEED, se ocupa el generador del kernel de linux (`/dev/random` en modo bloqueante y `/dev/urandom` en modo no bloqueante).

5.2. Programa para generar TOKENS

A continuación se explican las partes más importantes del programa: las implementaciones de los algoritmos tokenizadores mostrados en la sección 3.2.

En un intento por no complicar demasiado la exposición del programa, se redujo al máximo la aparición de código en el documento. Aquí solo se muestran las implementaciones de los algoritmos tokenizadores, y aún en estos casos, solamente se muestran las fracciones más relevantes. Al día de hoy, el programa cuenta con alrededor de 16000 líneas de código (contando comentarios y líneas en blanco), por lo que cualquier intento de incluir aquí a todo el programa resultaría en un documento de dimensiones estratosféricas. Si lo que se busca es una implementación en específico que no aparezca aquí, o mayor detalle en el código, se puede consultar la documentación en línea: https://rqf7.github.io/PROYECTO_LOVELACE/DOCUMENTACION_DOXYGEN/HTML/INDEX.html. Durante el desarrollo del programa se han hecho muchos esfuerzos para mantenerla completa.

Antes de iniciar con la explicación sobre las implementaciones sería conveniente aclarar un par de puntos sobre las reglas que sigue el formato del código, para facilitar la lectura: todos los identificadores se encuentran en español (el conjunto permitido por los caracteres AMERICAN STANDARD CODE FOR INFORMATION INTERCHANGE (ASCII)); las clases y espacios de nombres siguen **EsteFormato** mientras que las variables y funciones siguen **esteOtroFormato**; las variables miembro de una clase llevan siempre en el nombre una «m» como prefijo. Se pueden encontrar mayores detalles en: https://rqf7.github.io/PROYECTO_LOVELACE/REGLAS_DE_ESTILO.html.

5.2.1. Módulo de FFX

La operación FORMAT-PRESERVING FEISTEL-BASED ENCRYPTION (FFX) A10 se describió en la sección 3.2.1. La parte modular de su operación se encuentra en la función de ronda que ocupa la red Feistel interna (pseudocódigo 3.1); la implementación de esta función se muestra en el código fuente 1. FFX A10 funciona con una red Feistel alternante (sección 2.2.3), por lo que también es importante la implementación de este proceso; el código fuente 2 muestra la operación de cifrado y 3 muestra el descifrado.

Como se mostró en el diseño del programa (capítulo 4), la función de ronda implementa la interfaz de una función con inverso simétrico (diagrama 4.4), por lo que la función `operar` corresponde a la definida por la interfaz. Otro aspecto importante del código 1 es la primitiva criptográfica usada: una instancia de ADVANCED ENCRYPTION STANDARD (AES) CIPHER-BLOCK CHAINING (CBC) MESSAGE AUTHENTICATION CODE (MAC) de Crypto++. La operación de esa función se puede dividir en tres partes claramente distinguibles: el armado de la entrada a la primitiva (hasta la línea 160); la operación de la primitiva (hasta la línea 166); y por último, el formateo de la salida para la preservación del formato (el resto de la función).

Una red Feistel alternante es una red Feistel, la cuál a su vez implementa a una función con inverso; es por esto que los métodos mostrados en los códigos fuente 2 y 3 se llaman `operar` y `deoperar`, respectivamente. La red feistel alternante tiene dos funciones de ronda: una para las pares y otra para las impares. Estas son clases que tengan la forma de una función con inverso: la clase de la función de ronda de FFX A10, presentada en 1, implementa a una función con inverso simétrico (la cual a su vez implementa a una función con inverso). De esta forma se puede instanciar a FFX manteniendo totalmente desacoplado al código de las redes Feistel (ver ACOPLAMIENTO).

5.2.2. Módulo de BPS

El funcionamiento de BRIER-PEYRIN-STERN (BPS) se describe de forma más amplia en la sección 3.2.2.

A pesar de que se puede ver a BPS como una versión más específica de FORMAT-PRESERVING FEISTEL-BASED ENCRYPTION (FFX), esta implementación se hizo independiente de la de FFX.

De forma general, el módulo de BPS consta de los 2 métodos principales, el de cifrar y el de descifrar, los cuales son la implementación del modo de operación utilizado por BPS y que se describe en la sección 3.2.2. Estos métodos se muestran en los códigos 4, 5, 6 y 7.

Otro parte crucial de BPS es el cifrado interno BC, por lo cual es importante mostrar su implementación. Los códigos 8 y 9 están basados en el pseudocódigo para el cifrado BC descrito en 3.2.2.

Como se observa en el código 9, el cifrado interno usa a su vez un cifrado de ronda, que es un cifrado por bloques (en este caso ADVANCED ENCRYPTION STANDARD (AES) o DATA ENCRYPTION STANDARD (DES)), el cual se hizo usando la librería de Crypto++.

```
158  /**
159   * Implementación de función de ronda con CBC-MAC-AES.
160   *
161   * \return Texto cifrado de la longitud necesaria.
162   */
163
164  template<typename tipo>
165  Arreglo<tipo> RondaFFXA10<tipo>::operar(
166      const std::vector<Arreglo<tipo>> &textoEnClaro    /**< Texto a cifrar. */
167  )
168  {
169      /* Tweak. */
170      for (unsigned int i = 0, j = 8; i < mLongitudTweak; i++, j++)
171          mEntrada[j] = mTweak[i];
172
173      /* Representación numérica de mensaje. */
174      entero representacionNumero = convertirANumero<tipo, entero>(
175          textoEnClaro[0], mRadix);
176      for (int i = 0; i < 8; i++)
177          mEntrada[7 + mLongitudTweak + i + 1] =
178              static_cast<unsigned char>(representacionNumero >> (8 + i));
179
180      /* Generar MAC */
181      CryptoPP::CBC_MAC<CryptoPP::AES> cbcmac {mLlave};
182      cbcmac.Update(mEntrada, mLongitudEntrada);
183      unsigned char mac[cbcmac.DigestSize()];
184      cbcmac.TruncatedFinal(mac, cbcmac.DigestSize());
185
186      /* Partir por mitad */
187      Arreglo<int> ladoIzquierdo (8), ladoDerecho(8);
188      for (int i = 0; i < 16; i++)
189          if (i < 8)
190              ladoIzquierdo[i] = mac[i];
191          else
192              ladoDerecho[i - 8] = mac[i];
193
194      /* Formatear salida a longitud adecuada. */
195      entero numeroIzquierdo = convertirANumero<int, entero>(ladoIzquierdo, 256);
196      entero numeroDerecho = convertirANumero<int, entero>(ladoDerecho, 256);
197      entero z;
198      if (mLongitud <= 9)
199          z = modulo(numeroDerecho, potencia<entero>(mRadix, mLongitud));
200      else
201          z = (modulo(numeroIzquierdo, potencia<entero>(mRadix, mLongitud - 9))
202              * potencia<entero>(mRadix, 9))
203              + modulo(numeroDerecho, potencia<entero>(mRadix, 9));
204
205      return convertirAArreglo<tipo, entero>(z, mRadix, mLongitud);
206  }
```

Código fuente 1: Función de ronda de FFX

IMPLEMENTACIONES/FFX/CABECERAS/RONDA_FFX_A10.HH

```

146 /**
147 * Realiza todas las rondas del proceso de cifrado. La operación de una
148 * ronda se resume como:
149 * Si es par,  $PI_{\{i\}} = F(PD_{\{i - 1\}}) \text{ xor } PI_{\{i - 1\}}$ 
150 *  $PD_{\{i\}} = PD_{\{i - 1\}}$ 
151 * Si es impar,  $PD_{\{i\}} = F(PI_{\{i - 1\}}) \text{ xor } PD_{\{i - 1\}}$ 
152 *  $PI_{\{i\}} = PI_{\{i - 1\}}$ 
153 * La función de ronda usada depende de si esta es par, o impar.
154 *
155 * \return Bloque cifrado.
156 */
157
158 template <typename tipo>
159 Arreglo<tipo> RedFeistelAlternante<tipo>::operar(
160     const std::vector<Arreglo<tipo>>& textoEnClaro      /*< Bloque a cifrar. */
161 )
162 {
163     Arreglo<Arreglo<tipo>> partes = textoEnClaro[0] / Arreglo<unsigned int>{
164         (textoEnClaro[0].obtenerNumeroDeElementos() / 2) + mDesbalanceo};
165     for (mRondaActual = 0; mRondaActual < mNumeroDeRondas; mRondaActual++)
166     {
167         if (mRondaActual % 2 == 0)
168             partes[0] = std::move(mOperadorSuma->operar(
169                 {partes[0], mFuncionDeRonda->operar({partes[1]})}));
170         else
171             partes[1] = std::move(mOperadorSuma->operar(
172                 {partes[1], mFuncionDeRondaImpar->operar({partes[0]})}));
173     }
174     return static_cast<Arreglo<int>>(partes[0])
175     || static_cast<Arreglo<int>>(partes[1]);
176 }
```

Código fuente 2: Cifrado con red Feistel alternante

IMPLEMENTACIONES/REDES_FEISTEL/CABECERAS/RED_FEISTEL_ALTERNANTE.HH

```

178 /**
179 * Realiza todas las rondas del proceso de descifrado. La operación de las
180 * rondas es la misma que en el proceso de cifrado; no se hacen una sola
181 * función porque es necesario mantener la estructura dada por la superclase
182 * y porque es necesario llevar el contador de rondas en orden descendente.
183 *
184 * \return Bloque descifrado.
185 *
186 * \sa http://www.cplusplus.com/reference/utility/move/
187 */
188
189 template <typename tipo>
190 Arreglo<tipo> RedFeistelAlternante<tipo>::deoperar(
191     const std::vector<Arreglo<tipo>>& textoCifrado    /*< Bloque a descifrar. */)
192 )
193 {
194     Arreglo<Arreglo<tipo>> partes = textoCifrado[0] / Arreglo<unsigned int>{
195         (textoCifrado[0].obtenerNumeroDeElementos() / 2) + mDesbalanceo};
196     for (mRondaActual = mNumeroDeRondas - 1; mRondaActual >= 0; mRondaActual--)
197     {
198         if (mRondaActual % 2 == 0)
199             partes[0] = std::move(mOperadorSuma->deoperar({
200                 partes[0], mFuncionDeRonda->operar({partes[1]})));
201         else
202             partes[1] = std::move(mOperadorSuma->deoperar({
203                 partes[1], mFuncionDeRondaImpar->operar({partes[0]})));
204         if (mRondaActual == 0)
205             break;
206     }
207     return static_cast<Arreglo<int>>(partes[0])
208     | static_cast<Arreglo<int>>(partes[1]);
209 }
```

Código fuente 3: Descifrado con red Feistel alternante

IMPLEMENTACIONES/REDES_FEISTEL/CABECERAS/RED_FEISTEL_ALTERNANTE.HH

```

77 /**
78 * Este método sirve para cifrar la cadena dada con la llave y el tweak dados.
79 * El funcionamiento de este método es el del modo de operación del algoritmo
80 * de cifrado que preserva el formato BPS.
81 */
82
83
84 string CifradorBPS::cifrar(string mensaje, byte llave[], mpz_class tweak)
85 {
86     CifradorBC BC;
87     BC.colocarTipoCifrador(mTipoCifrador);
88     int tamCifradorDeRonda = BC.obtenerCifradorDeRonda().obtenerTamBloque();
89
90     /* Obtención del tamaño máximo de bloque del cifrado interno BC */
91     unsigned int tamTotal, tamMax;
92     tamTotal = mensaje.size();
93     tamMax = 2 * ((tamCifradorDeRonda - 32) * log(2)) / log(mAlfabeto.size());
94
95     /* Configuración del cifrado interno BC */
96     BC.colocarAlfabeto(mAlfabeto);
97     BC.colocarTamBloque(tamMax);
98     BC.colocarNumRondas(mNumRondas);
99
100    /* En caso de que la cadena dada tenga una longitud menor al tamaño máximo
101       del cifrado BC, simplemente se cifra la cadena con BC */
102    if(tamTotal <= tamMax)
103    {
104        BC.colocarTamBloque(tamTotal);
105        return BC.cifrar(mensaje, llave, tweak);
106    }
107
108    string bloqueA {" "};
109    string salida {" "};
110    string salidaFinal {" "};
111
112    unsigned int numBloques = tamTotal / tamMax;
113    unsigned int tamUltimoBloque = tamTotal % tamMax;
114    mpz_class contadorU = 0;
115    mpz_class tweakU = 0;
116

```

Código fuente 4: Función de cifrado de BPS (parte 1).

IMPLEMENTACIONES/BPS/CIFRADOR_BPS.CPP

```
116
117 /* En caso de que la cadena dada tenga una longitud mayor al tamaño máximo
118 del cifrado BC, se van cifrando bloques x de longitud igual a la longitud
119 máxima de BC, donde x es igual a la suma modular de un bloque i con un
120 bloque i-1 */
121 BC.colocarTamBloque(tamMax);
122 for (unsigned int i = 0; i < numBloques; i++)
123 {
124     /* Xor del contador u = 2^16 + 2^48 con el tweak que entrara
125     al cifrado BC */
126     contadorU = i;
127     contadorU = (contadorU << 16) + (contadorU << 48);
128     tweakU = tweak ^ contadorU;
129
130     /* Cifrado de la suma modular del bloque i con el bloque i-1 */
131     bloqueA = mensaje.substr(i * tamMax, tamMax);
132     salida = BC.cifrar(mCodificador.sumaMod(bloqueA, salida), llave, tweakU);
133     salidaFinal += salida;
134 }
135
136 /* En caso de que la cadena dada tenga una longitud que no es múltiplo
137 del tamaño máximo de BC, el último bloque se cifra configurando BC
138 a su longitud */
139 if(tamUltimoBloque != 0)
140 {
141     /* Xor del contador u = 2^16 + 2^48 con el tweak que entrará
142     al cifrado BC */
143     BC.colocarTamBloque(tamUltimoBloque);
144     contadorU = numBloques;
145     contadorU = (contadorU << 16) + (contadorU << 48);
146     tweakU = tweak ^ contadorU;
147
148     /* Cifrado de la suma modular del bloque i con el bloque i-1 */
149     bloqueA = mensaje.substr(tamTotal - tamUltimoBloque, tamUltimoBloque);
150     salida = BC.cifrar(mCodificador.sumaMod(bloqueA, salida), llave, tweakU);
151     salidaFinal += salida;
152 }
153
154 return salidaFinal;
155 }
```

Código fuente 5: Función de cifrado de BPS (parte 2).

IMPLEMENTACIONES/BPS/CIFRADOR_BPS.CPP

```
158
159 /**
160 * Este método sirve para descifrar la cadena dada con la llave y tweak dados.
161 * El funcionamiento de este método es el del modo de operación del algoritmo
162 * de cifrado que preserva el formato BPS.
163 */
164
165 string CifradorBPS::descifrar(string mensaje, byte llave[], mpz_class tweak)
166 {
167     CifradorBC BC;
168     BC.colocarTipoCifrador(mTipoCifrador);
169     int tamCifradorDeRonda = BC.obtenerCifradorDeRonda().obtenerTamBloque();
170
171     /* Obtención del tamaño máximo de bloque del cifrado interno BC */
172     unsigned int tamTotal, tamMax;
173     tamTotal = mensaje.size();
174     tamMax = 2 * ((tamCifradorDeRonda - 32) * log(2)) / log(mAlfabeto.size());
175
176     /* Configuración del cifrado interno BC */
177     BC.colocarAlfabeto(mAlfabeto);
178     BC.colocarTamBloque(tamMax);
179     BC.colocarNumRondas(mNumRondas);
180
181     /* En caso de que la cadena dada tenga una longitud menor al tamaño máximo
182     del cifrado BC, simplemente se descifra la cadena con BC */
183     if(tamTotal <= tamMax)
184     {
185         BC.colocarTamBloque(tamTotal);
186         return BC.descifrar(mensaje, llave, tweak);
187     }
188
189     string bloqueA {" "};
190     string bloqueB {" "};
191     string salida {" "};
192     string salidaFinal {" "};
193
194     unsigned int numBloques = tamTotal / tamMax;
195     unsigned int tamUltimoBloque = tamTotal % tamMax;
196     mpz_class contadorU = 0;
197     mpz_class tweakU = 0;
198
```

Código fuente 6: Función de descifrado de BPS (parte 1).

IMPLEMENTACIONES/BPS/CIFRADOR_BPS.CPP

```
198
199 /* En caso de que la cadena dada tenga una longitud que no es múltiplo
200 del tamaño máximo de BC, el último bloque se descifra configurando BC
201 a su longitud */
202 if(tamUltimoBloque != 0)
203 {
204     /* Xor del contador u = 2^16 + 2^48 con el tweak que entrara
205     al cifrado BC */
206     BC.colocarTamBloque(tamUltimoBloque);
207     contadorU = numBloques;
208     contadorU = (contadorU << 16) + (contadorU << 48);
209     tweakU = tweak ^ contadorU;
210
211     /* Obtención del bloque i e i-1 */
212     bloqueB = mensaje.substr(tamTotal - tamUltimoBloque, tamUltimoBloque);
213     bloqueA = mensaje.substr((numBloques - 1) * tamMax, tamMax);
214
215     /* Resta modular del resultado de decifrar el bloque i, menor el bloque i-1 */
216     salida = mCodificador.restaMod(BC.descifrar(bloqueB, llave, tweakU), bloqueA);
217     salidaFinal = salida + salidaFinal;
218 }
219
220 /* En caso de que la cadena dada tenga una longitud mayor al tamaño máximo
221 del cifrado BC, se van obteniendo bloques descifrados x de longitud igual
222 a la longitud máxima de BC, donde x es igual a la resta modular del
223 resultado de descifrar un bloque un bloque i con un bloque i-1 */
224 BC.colocarTamBloque(tamMax);
225 for(int i = numBloques - 1; i > 0; i--)
226 {
227     /* Xor del contador u = 2^16 + 2^48 con el tweak que entrara
228     al cifrado BC */
229     contadorU = i;
230     contadorU = (contadorU << 16) + (contadorU << 48);
231     tweakU = tweak ^ contadorU;
232
233     /* Obtención del bloque i e i-1 */
234     bloqueB = mensaje.substr(i * tamMax, tamMax);
235     bloqueA = mensaje.substr((i - 1) * tamMax, tamMax);
236
237     /* Resta modular del resultado de decifrar el bloque i, menor el bloque i-1 */
238     salida = mCodificador.restaMod(BC.descifrar(bloqueB, llave, tweakU), bloqueA);
239     salidaFinal = salida + salidaFinal;
240 }
241
242 /* Descifrado del primer bloque en caso de que la cadena dada fuese de una
243 longitud mayor al tamaño máximo del cifrado BC */
244 bloqueA = mensaje.substr(0, tamMax);
245 salida = BC.descifrar(bloqueA, llave, tweak);
246 salidaFinal = salida + salidaFinal;
247
248 return salidaFinal;
249 }
```

Código fuente 7: Función de descifrado de BPS (parte 2).

IMPLEMENTACIONES/BPS/CIFRADOR_BPS.CPP

```
111
112 /**
113 * Este método sirve para cifrar la cadena dada con la llave y el tweak dados.
114 * El funcionamiento de este método es el del cifrado interno BC del
115 * algoritmo de cifrado que preserva el formato BPS.
116 */
117
118 string CifradorBC::cifrar(string mensaje, byte llave[], mpz_class tweak)
119 {
120     /* Mensaje de error para cuando la cadena dada es de una longitud
121     distinta al tamaño de bloque */
122     if(mensaje.size() != mTamBloque)
123     {
124         cout << "ERROR, cadena de longitud distinta a la establecida." ;
125         cout << endl;
126         exit(-1);
127     }
128
129     UtilidadesBPS util;
130     unsigned int l, r;
131     string mensajeIzq{""};
132     string mensajeDer{""};
133
134     /* Obtención de las longitudes de la parte izquierda (l) y derecha (r)
135     de la cadena dada para cifrar */
136     l = (mTamBloque%2 == 0) ? mTamBloque/2 : (mTamBloque+1)/2;
137     r = (mTamBloque%2 == 0) ? mTamBloque/2 : (mTamBloque-1)/2;
138
139     /* Construcción de la parte izquierda y derecha del la cadena dada */
140     for(unsigned int i=0; i<mTamBloque; i++)
141         if(i<l) mensajeIzq += mensaje[i];
142         else     mensajeDer += mensaje[i];
143
144     /* Obtención de los valores de composición de la parte izquierda y derecha */
145     mpz_class ladoIzq = componer(mensajeIzq);
146     mpz_class ladoDer = componer(mensajeDer);
147
148     /* Obtención los subtweaks izquierdo y derecho */
149     mpz_class tweakDer = tweak & (0xFFFFFFFF);
150     mpz_class tweakIzq = (tweak >> 32) & (0xFFFFFFFF);
151
152     mpz_class tweakDerAux = 0;
153     mpz_class tweakIzqAux = 0;
154     mpz_class entradaCr    = 0;
155     mpz_class salidaCr    = 0;
156     mpz_class maxS1       = 0;
157     mpz_class maxSR       = 0;
158     string entradaCrStr  {""};
159     string salidaCrStr   {""};
160
161     /* Asignación de s^l y s^r en las variables maxS1 y maxSR */
162     mpz_ui_pow_ui(maxS1.get_mpz_t(), mCardinalidad, l);
163     mpz_ui_pow_ui(maxSR.get_mpz_t(), mCardinalidad, r);
164     int tamCifradorDeRonda = mCifradorDeRonda.obtenerTamBloque();
165
```

Código fuente 8: Cifrado interno BC de BPS (parte 1).

IMPLEMENTACIONES/BPS/CIFRADOR_BC.CPP

```
165  /* Ciclo de la red Feistel */
166  for (unsigned int i = 0; i < mNumRondas; i++)
167  {
168      if (i % 2 == 0)
169      {
170          /* Corrimiento a la izquierda de f-32 bits del subtweak derecho
171          xor con el contador, donde f es el tamaño de bloque del cifrado
172          de ronda */
173          tweakDerAux = (tweakDer ^ i) << (tamCifradorDeRonda - 32);
174
175          /* Suma del subtweak recorrido con el lado derecho */
176          entradaCr = tweakDerAux + ladoDer;
177          entradaCrStr = util.numeroACadena(entradaCr);
178
179          /* Cifrado de la suma anterior */
180          salidaCrStr = mCifradorDeRonda.cifrar(entradaCrStr, llave);
181          salidaCr = util.cadenaANumero(salidaCrStr);
182
183          /* Suma modular del lado izquierdo con el resultado del
184          cifrado anterior */
185          ladoIzq = util.mod(ladoIzq, maxS1) + util.mod(salidaCr, maxS1);
186          ladoIzq = util.mod(ladoIzq, maxS1);
187      }
188  else
189  {
190      /* Corrimiento a la izquierda de f-32 bits del subtweak izquierdo
191      xor con el contador, donde f es el tamaño de bloque del cifrado
192      de ronda */
193      tweakIzqAux = (tweakIzq ^ i) << (tamCifradorDeRonda - 32);
194
195      /* Suma del subtweak recorrido con el lado izquierdo */
196      entradaCr = tweakIzqAux + ladoIzq;
197      entradaCrStr = util.numeroACadena(entradaCr);
198
199      /* Cifrado de la suma anterior */
200      salidaCrStr = mCifradorDeRonda.cifrar(entradaCrStr, llave);
201      salidaCr = util.cadenaANumero(salidaCrStr);
202
203      /* Suma modular del lado derecho con el resultado del
204      cifrado anterior */
205      ladoDer = util.mod(ladoDer, maxSR) + util.mod(salidaCr, maxSR);
206      ladoDer = util.mod(ladoDer, maxSR);
207  }
208
209 }
210
211 /* Descomposición del lado izquierdo y derecho para concatenarlas y
212 obtener la cadena cifrada */
213 string mensajeCifrado;
214 mensajeCifrado += descomponer(ladoIzq,l);
215 mensajeCifrado += descomponer(ladoDer,r);
216 return mensajeCifrado;
217 }
218 }
```

Código fuente 9: Cifrado interno BC de BPS (parte 2).

IMPLEMENTACIONES/BPS/CIFRADOR_BC.CPP

5.2.3. Módulo de TKR

La descripción de los algoritmos que componen a este método se hizo en la sección 3.2.3. En la sección 4.2.3 se mostró la vista estática que tiene este módulo. En esta sección se muestran los aspectos más importantes de su implementación.

```
42 /**
43 * Proceso para generar un token a partir de un número de tarjeta. Primero
44 * se busca en la base de datos, si ya hay un token para el número dado, es
45 * este el que se regresa; sino, se crea uno (y se inserta en la base) con la
46 * función pseudoaleatoria.
47 *
48 * \return Token asociado al PAN dado.
49 */
50
51 ArregloDeDigitos TKR::tokenizar(
52     const ArregloDeDigitos& pan           /*< Número de tarjeta. */
53 )
54 {
55     Registro informacion = mBaseDeDatos->buscarPorPan(pan);
56     if (informacion.obtenerToken() == Arreglo<int>{})
57     {
58         auto division = pan
59             / Arreglo<unsigned int>{6, pan.obtenerNumeroDeElementos() - 1};
60         ArregloDeDigitos temporal =
61             mFuncionPseudoaleatoria->operar(
62                 static_cast<ArregloDeDigitos>(division[1]).obtenerNumeroDeElementos());
63         temporal = static_cast<ArregloDeDigitos>(division[0]) || temporal;
64         informacion.colocarToken(temporal
65             || ArregloDeDigitos{modulo(algoritmoDeLuhn(temporal) + 1, 10)});
66         informacion.colocarPAN(pan);
67     try
68     {
69         mBaseDeDatos->guardar(informacion);
70     } catch (sql::SQLException& error)
71     {
72         return this->tokenizar(pan);
73     }
74 }
75 return informacion.obtenerToken();
76 }
```

Código fuente 10: Proceso de cifrado de TKR.

IMPLEMENTACIONES/TKR/TKR.CPP

Los códigos fuente 10 y 11 muestran los procesos de cifrado y descifrado, respectivamente. Estas operaciones están descritas por los pseudocódigos 3.5 y 3.6. La operación de cifrado depende de una función psudoaleatoria; dentro del contexto de la clase TKR, esta es cualquier otra clase que implemente a una función que recibe enteros y regresa arreglos de dígitos. La operación de descifrado es simplemente una consulta en la base de datos; en caso de una búsqueda infructuosa, se lanza una excepción.

Como se muestra en el diagrama 4.6, TKR implementa la interfaz de un algoritmo tokenizador irreversible (que a su vez implementa a un algoritmo tokenizador), por lo que los métodos mostrados en los

```

78 /**
79 * Proceso de detokenización. Simplemente busca en la base de datos el
80 * PAN asociado al token dado.
81 *
82 * \return PAN asociado al token dado.
83 */
84
85 ArregloDeDigitos TKR::detokenizar(
86     const ArregloDeDigitos& token           /* Token (generado previamente). */
87 )
88 {
89     Registro informacion = mBaseDeDatos->buscarPorToken(token);
90     if (informacion.obtenerPAN() == Arreglo<int>{})
91         throw TokenInexistente{"El token no está en la base de datos."};
92     return informacion.obtenerPAN();
93 }

```

Código fuente 11: Proceso de descifrado de TKR.

IMPLEMENTACIONES/TKR/TKR.CPP

códigos fuente 10 y 11 son los definidos por la interfaz. Al tratarse de un método irreversible, ambos procesos (tokenización y detokenización) reciben el PERSONAL ACCOUNT NUMBER (PAN) o el TOKEN completos, según sea el caso.

```

53 /**
54 * Utiliza la función interna para generar una cadena de bits pseudoaleatoria;
55 * después interpreta esa cadena con los datos miembro (la cardinalidad y
56 * la longitud de la cadena) para regresar la cadena con el formato adecuado.
57 *
58 * \return Arreglo pseudoaleatorio con \ref mLongitudDeBits de longitud.
59 */
60
61 ArregloDeDigitos FuncionRN::operar(const vector<unsigned int>& entrada)
62 {
63     Arreglo<unsigned char> binarioAleatorio =
64         mFuncionInterna->operar({mLongitudDeCadena, mContador});
65     ArregloDeDigitos resultado (mLongitudDeCadena);
66     for (unsigned int i = 0; i < mLongitudDeBits; i++)
67         resultado[i] = binarioAleatorio[i] % 10;
68     mContador++;
69     return resultado;
70 }

```

Código fuente 12: Función RN.

IMPLEMENTACIONES/TKR/FUNCION_RN.CPP

La función del código fuente 12 es la implementación del pseudocódigo descrito en 3.7. Esta clase implementa la interfaz de una función que recibe enteros y regresa arreglos de dígitos (siguiendo el modelo de débil acomplamiento dado por las interfaces de las funciones). Básicamente, se trata de la transformación de los bytes aleatorios dados por la función interna en el número de dígitos pedidos. En el contexto de esta clase, la función interna tiene el mismo aspecto: una función que recibe enteros y regresa arreglos de bytes.

```
51 /**
52 * Operación de función: recibe en el vector de los argumentos un
53 * contador (el estado el algoritmo en RN) y la longitud de bits que el
54 * resultado debe tener. La longitud real regresada es el múltiplo del
55 * tamaño de bloque de AES (128 bits) más cercano hacia arriba.
56 *
57 * Este es el algoritmo propuesto en el artículo de TKR para instanciar f
58 * con un cifrado por bloques.
59 *
60 * \return Arreglo de bytes con contenido del cifrado.
61 */
62
63 Arreglo<unsigned char> PseudoaleatorioAES::operar(
64     const std::vector<entero>& entrada    /*< Contador y longitud de resultado. */
65 )
66 {
67     unsigned int numeroDeBloques = static_cast<unsigned int>(ceil(entrada[0] / 128.0));
68     Arreglo<unsigned char> resultado (numeroDeBloques * 16);
69     unsigned char bufferUno[16];
70     memset(bufferUno, 0, 16);
71     for (int i = 0; i < 8; i++)
72         bufferUno[i] = static_cast<unsigned char>(entrada[1] >> (i * 8));
73     unsigned char bufferDos[16];
74     cifrar(bufferUno, bufferDos, 0);
75     for (unsigned int i = 0; i < numeroDeBloques; i++)
76     {
77         cifrar(bufferDos, bufferUno, i);
78         for (int j = 0; j < 16; j++)
79             resultado[(i * 16) + j] = bufferUno[j];
80     }
81     return resultado;
82 }
```

Código fuente 13: Generación de bytes pseudoaleatorios basado en AES.

IMPLEMENTACIONES/TKR/PSEUDOALEATORIO_AES.CPP

Por último, el código fuente 13 muestra cómo se puede ocupar un cifrado de bloques (ADVANCED ENCRYPTION STANDARD (AES) de 128 bits, en este caso) para producir arreglos de bytes con aspecto aleatorio. La operación de esta función es análoga al modo de operación COUNTER MODE (CTR) (sección 2.2.6)

5.2.4. Módulo de AHR

En esta sección se explica, de manera general, la implementación del algoritmo AHR, descrito en la sección 3.2.4.

Partiendo de lo general hacia lo particular, en la sección de código 14 se ve el método *principal*; primero, dependiendo de la longitud del PERSONAL ACCOUNT NUMBER (PAN) calcula los límites entre los cuales el TOKEN será válido. Posteriormente, mediante el método *crearBloqueT*, descrito en 15, se concatena a la salida de la función SHA256 (que recibe como entrada el ISSUER IDENTIFICATION NUMBER (INN)), la representación binaria de la entrada *X*, que es el número de cuenta que se va a tokenizar. Una vez que se tiene el bloque *T*, se cifra mediante ADVANCED ENCRYPTION STANDARD (AES) con una llave de 256 bits y se obtiene la representación decimal de los últimos bits del bloque cifrado; el resultado de esta operación es el TOKEN candidato, pues aún falta revisar si el TOKEN se encuentra dentro de los límites obtenidos al principio; en caso de que no cumpla, se usa CIFRADO DE CAMINATA CÍCLICA hasta obtener un TOKEN válido. Finalmente, se revisa en la base de datos si el TOKEN creado existe (este algoritmo permite generar varios TOKENS para un mismo PAN): si no existe, guarda la relación PAN-TOKEN en la base de datos; si sí, aumenta en uno el INN y vuelve a ejecutar el algoritmo desde el inicio. Respecto a la detokenización, como se observa en 16, se realiza directamente una búsqueda en la base de datos, en caso de no encontrar el TOKEN ingresado, lanza un error indicando que no existe en la base de datos.

5.2.5. Módulo de DRBG

En la sección B se resume el estándar del NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY (NIST) para la creación de un DETERMINISTIC RANDOM BIT GENERATOR (DRBG). En 3.2.5 se muestran dos posibles implementaciones concretas: una basada en una función hash y la otra basada en un cifrado por bloques. En 4.2.5 se explica la distribución de las clases creada para este módulo. En esta sección se muestran los aspectos más significativos de las implementaciones.

Como se muestra en el diagrama 4.8, un generador solamente expone una función pública: *operar* para la generación de bytes; el nombre viene dado por el contrato con la interfaz de una función que recibe enteros y regresa arreglos de bytes. El código fuente 17 muestra esta implementación, la cuál depende de *generarBytes*, que dentro del contexto de un DRBG genérico, es abstracta.

De acuerdo a lo explicado en la sección 5.1.3, el comportamiento por defecto del constructor de un

```
284 /**
285 * Método principal que se encarga de crear el token:
286 * 1. Crea el mBloqueT al concatenar la salida truncada del SHA256 (alimentada
287 * con mEntradaU) y la representación binaria de mEntradaX.
288 * 2. Realiza al menos una vez los siguientes pasos:
289 *   2.1 Cifrar el mBloqueT mediante AES256 y lo guarda en el mBloqueC.
290 *   2.2 Obtiene la representación decimal de los últimos mLongitudBytesEntradaX
291 *       bits del mBloqueC y los guarda en token.
292 *   2.3 Comprueba que el valor en token sea menor a 10^mNumeroDigitosEntradaX:
293 *       en otras palabras, verifica que tenga mNumeroDigitosEntradaX dígitos.
294 *       En caso de que no lo tenga, usa el método de la caminata cíclica al
295 *       regresar al paso 2.1, pero cifrando el mBloqueC actual.
296 *   3. Verifica si existe el token en la base de datos. Si existe, regresa al
297 *       paso 1, con la misma llave y la misma entrada mEntradaX, pero aumenta
298 *       en uno la entrada mEntradaU.
299 */
300 void AHR::tokenizarHibridamente()
301 {
302     entero limiteS = potencia<entero>(10, mNumeroDigitosEntradaX);
303     entero limiteI = potencia<entero>(10, (mNumeroDigitosEntradaX-1)) - 1;
304
305     /* Primer paso del algoritmo. */
306     crearBloqueT();
307
308     /* Caminata cíclica para obtener un token dentro del dominio del mensaje. */
309     do{
310         /* Segundo paso del algoritmo: cifrar el mBloqueT y ponerlo en mBloqueC. */
311         mCifrador.cifrarBloque(mBloqueT);
312         mBloqueC = mCifrador.obtenerBloqueTCifrado();
313
314         /* Tercer y cuarto paso del algoritmo: verificar que esté dentro del dominio
315          * del mensaje y pasarlo a su forma decimal.
316         */
317         obtenerNumeroC();
318         memcpy(mBloqueT, mBloqueC, M);
319     }while(mToken >= limiteS || mToken <= limiteI);
320
321     /* Quinto paso del algoritmo: revisar si existe el token generado.*/
322     if(existeToken())
323     {
324         /* Aumentar en uno la mEntradaU y volver a correr el algoritmo.*/
325         mEntradaU += 1;
326         tokenizarHibridamente();
327     }
328 }
```

Código fuente 14: Tokenización mediante AHR.

IMPLEMENTACIONES/AHR/AHR.CPP

```

193 /**
194 * Este método se encarga del primer paso del algoritmo: obtener el mBloqueT
195 * que será utilizado como entrada para el cifrado por bloques. Primero
196 * obtiene la salida de la función pública (SHA256 en este caso) de la entrada
197 * adicional mEntradaU. Luego llama a la función obtenerBitsX para llenar los
198 * últimos mLongitudBytesEntradaX bloques del mBloqueT y, finalmente, le
199 * concatena al inicio los bytes más significativos de la salida del SHA para
200 * completar el tamaño de bloque del cifrado.
201 */
202 void AHR::crearBloqueT()
203 {
204     string cadenaU = to_string(mEntradaU);
205     SHA256 funcionF;
206     byte salidaFCompleta[SHA256::DIGESTSIZE];
207
208     /* Obtener valor SHA256 de la entrada adicional*/
209     funcionF.Update(
210         reinterpret_cast<const unsigned char*>(cadenaU.c_str()),
211         cadenaU.length()
212     );
213     funcionF.Final(salidaFCompleta);
214
215     obtenerBitsX();
216
217     int j = 0;
218     for(int i=0; i < M - mLongitudBytesEntradaX; i++)
219     {
220         mBloqueT[i] = salidaFCompleta[j];
221         j++;
222     }
223 }
```

Código fuente 15: Primer paso para la tokenización con AHR.

IMPLEMENTACIONES/AHR/AHR.CPP

```

360 /**
361 * Este método se encarga de regresar el PAN que corresponde al
362 * token dado en el argumento. Como este es un algoritmo irreversible,
363 * busca en la base de datos el token dado y regresa el PAN asociado.
364 * En caso de no existir, lanza una excepción.
365 */
366 ArregloDeDigitos AHR::detokenizar(const ArregloDeDigitos& token)
367 {
368     Registro informacion = mAccesoADatos->buscarPorToken(token);
369     if (informacion.obtenerPAN() == Arreglo<int>{})
370         throw TokenInexistente("El token no está en la base de datos.");
371     return informacion.obtenerPAN();
372 }
```

Código fuente 16: Primer paso para la detokenización con AHR.

IMPLEMENTACIONES/AHR/AHR.CPP

```
66 if (mFuenteDeAlatoriedad == nullptr)
67 {
68     mBanderaDeRecursos = true;
69     try
70     {
71         mFuenteDeAlatoriedad = new AleatoriedadHardware{};
72     }
73     catch(CryptoPP::RDSEED_Error &error)
74     {
75         mFuenteDeAlatoriedad = new AleatoriedadTrivial{};
76     }
77 }
78
79 mSemilla = mFuenteDeAlatoriedad->operar({mLongitudSemilla});
80 }
81
82 /**
83 * Libera la memoria que se reservó en el constructor. Si el apuntador de la
84 * fuente de aleatoriedad es externo, no se libera.
85 */
86
87 DRBG::~DRBG()
88 {
89     if (mBanderaDeRecursos)
90         delete mFuenteDeAlatoriedad;
```

Código fuente 17: Función pública de generadores pseudoaleatorios.

IMPLEMENTACIONES/DRBG/DRBG.CPP

DRBG es determinar, en tiempo de ejecución, si el procesador actual soporta la instrucción de ensamblador RD-SEED. En caso positivo, la fuente de entropía es una instancia de la clase de aleatoriedad por hardware. En caso negativo, la entropía es una instancia de la clase de aleatoriedad trivial. También existe la posibilidad de que el constructor reciba una fuente de aleatoriedad creada con anterioridad; en este caso, el DRBG no es responsable de la gestión de la memoria. Esta segunda característica permite instanciar varios generadores con una misma fuente de aleatoriedad.

En los códigos fuente 18 y 19 se muestran las dos operaciones más importantes del generador basado en funciones hash: la generación de bytes y una función de generación interna. Como se puede ver en el primero de los códigos, la función de generación de bytes depende de la función interna: esta genera los bytes pedidos y después, la función de generación de bytes, se encarga de actualizar el estado interno.

En los códigos fuente 20 y 21 se muestran las dos operaciones más importantes del generador basado en un cifrado por bloques. La primera se trata de la función de generación de bytes; como se puede observar, su operación es idéntica al modo de operación de contador (sección 2.2.6). La segunda función actualiza el estado cada vez que hay un cambio de semilla.

```

117 /**
118 * Define el proceso de generación de bytes pseudoaleatorios. Corresponde a la
119 * sección 10.1.1.4 del estándar. El trabajo para generar los nuevos bytes se
120 * encuentra en funcionDeGeneracion (es la misma división del documento, pero
121 * con nombres distintos), también se genera un nuevo estado para la semilla.
122 *
123 * \return Arreglo con bytes aleatorios.
124 */
125
126
127 Arreglo<unsigned char> HashDRBG::generarBytes(
128     entero longitud           /*< Número de bytes deseados. */
129 )
130 {
131     auto resultado = funcionDeGeneracion(static_cast<unsigned int>(longitud));
132     auto temporal = hash(Arreglo<unsigned char>{3} || mSemilla);
133     mSemilla = mSemilla + temporal
134         + mConstanteSemilla + Arreglo<unsigned char>(mContadorDePeticiones);
135     return resultado;

```

Código fuente 18: Función de generación de bytes de hash DRBG.

IMPLEMENTACIONES/DRBG/HASH_DRBG.CPP

```

166 /**
167 * Función ocupada por generarBytes para generar el número de bytes aleatorios
168 * usando la función hash interna y el valor de la semilla.
169 */
170
171
172 Arreglo<unsigned char> HashDRBG::funcionDeGeneracion(
173     unsigned int longitudDeSalida      /*< Longitud de la salida. */
174 )
175 {
176     Arreglo<unsigned char> resultado;
177     unsigned int longitud = mFuncionHash->DigestSize();
178     unsigned int numeroDeBloques = ceil(
179         static_cast<double>(longitudDeSalida) / longitud);
180     Arreglo<unsigned char> datos {mSemilla};
181     for (unsigned int i = 0; i < numeroDeBloques; i++)
182     {
183         resultado = resultado || hash(datos);
184         datos = datos || Arreglo<unsigned char>{1};
185     }
186     return resultado.fragmentar(Arreglo<unsigned int>{longitudDeSalida})[0];

```

Código fuente 19: Función de generación interna de hash DRBG.

IMPLEMENTACIONES/DRBG/HASH_DRBG.CPP

```

82 /**
83  * Función generadora de bytes aleatorios (llamada desde clase
84  * abstracta).
85 *
86  * \return Arreglo con el número de bytes solicitados.
87 */
88
89 Arreglo<unsigned char> CTRDRBG::generarBytes(
90     entero longitud           /**< Longitud de salida. */
91 )
92 {
93     unsigned int longitudLocal = static_cast<unsigned int>(longitud);
94     Arreglo<unsigned char> resultado;
95     while (resultado.obtenerNúmeroDeElementos() < longitudLocal)
96     {
97         mSemilla = (mSemilla + 1ull).fragmentar(
98             Arreglo<unsigned int>{mLongitudBloque})[0];
99         resultado = resultado || cifrarBloque(mSemilla);
100    }
101 return resultado.fragmentar(Arreglo<unsigned int>{longitudLocal})[0];

```

Código fuente 20: Función de generación de bytes de CTR DRBG.

[IMPLEMENTACIONES/DRBG/CTR_DRBG.CPP](#)

```

103 /**
104  * Operación de actualización de estado. En el estándar corresponde a
105  * CTR_DRBG_Update (sección 10.2.1.2). Calcula un nuevo valor para
106  * la semilla y para la llave del cifrado por bloques.
107 */
108
109 void CTRDRBG::actualizarEstado(
110     const Arreglo<unsigned char>& entrada    /**< Arreglo de entrada (entropía). */
111 )
112 {
113     Arreglo<unsigned char> temporal;
114     while (temporal.obtenerNúmeroDeElementos() < mLongitudSemilla)
115     {
116         mSemilla = (mSemilla + 1ull).fragmentar(
117             Arreglo<unsigned int>{mLongitudBloque})[0];
118         temporal = temporal || cifrarBloque(mSemilla);
119     }
120     temporal = temporal.fragmentar(
121         Arreglo<unsigned int>{mLongitudSemilla})[0];
122     temporal = temporal ^ entrada;

```

Código fuente 21: Función de actualización de estado de CTR DRBG.

[IMPLEMENTACIONES/DRBG/CTR_DRBG.CPP](#)

5.3. Resultados

5.3.1. Resultados de las pruebas estadísticas a los DRBG

En esta sección se pretende demostrar la aleatoriedad de los DETERMINISTIC RANDOM BIT GENERATOR (DRBG) que fueron implementados, esto por medio del conjunto de pruebas del NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY (NIST), pruebas de las que se habla más a fondo en el apéndice C.

En las siguientes 6 tablas (5.1, 5.2, 5.3, 5.4, 5.5 y 5.6) se observan los resultados obtenidos para las 15 pruebas estadísticas descritas en [35] para 3 tipos distintos de DRBG (uno basado en ADVANCED ENCRYPTION STANDARD (AES) y otros dos basados en SECURE HASH ALGORITHM (SHA)) en cada uno de los niveles de seguridad que se tienen (112, 128, 192 y 256).

Para poder comprender la información más claramente, es necesario establecer los siguientes puntos:

- Las pruebas se realizaron con 10 millones de bits obtenidos por cada tipo de DRBG.
- Cada prueba realizada consta de 20 ejecuciones, y cada ejecución usa medio millón de bits.
- Se puede resumir que el valor de P va de 0 a 1, y que mientras más grande sea este valor, el DRBG probado es más aleatorio.
- Las pruebas de sumas acumulativas, de coincidencia sin superposición, de entropía aproximada, de excursiones aleatorias, y su variante constan de varias subpruebas.
- En las tablas, en las columnas de *proporción*, se pone el número de pruebas pasadas con respecto al número de pruebas (subpruebas si existen) que se hicieron en total.
- Posterior a las columnas de *proporción* se pone un ✓ en caso de que el DRBG aprobara de forma general la prueba o un ✗ en el caso contrario.

Pruebas estadísticas para el DRBG basado en cifrados por bloque (AES)						
Nivel de seguridad	112		128			
Prueba	Valor P	Proporción		Valor P	Proporción	
De frecuencia	0.991468	19/20	✓	0.437274	20/20	✓
De frecuencia en un bloque	0.017912	20/20	✓	0.048716	20/20	✓
De sumas acumulativas	0.872861	38/40	✓	0.356491	40/40	✓
De carreras	0.964295	20/20	✓	0.350485	20/20	✓
De la carrera más larga en un bloque	0.637119	20/20	✓	0.213309	20/20	✓
Del rango de matriz binaria	0.964295	20/20	✓	0.911413	20/20	✓

Continúa en siguiente página

<i>Continuación</i>						
Nivel de seguridad	112		128			
Prueba	Valor P	Proporción		Valor P	Proporción	
Espectral	0.637119	20/20	✓	0.213309	20/20	✓
De coincidencia sin superposición	0.475566	2925/2960	✓	0.513657	2930/2960	✓
De coincidencia con superposición	0.964295	20/20	✓	0.637119	20/20	✓
Estadística universal de Maurer	0.066882	20/20	✓	0.834308	19/20	✓
De entropía aproximada	0.739918	20/20	✓	0.834308	20/20	✓
De excusiones aleatorias	—	70/72	✓	0.485432	96/96	✓
Variante de excusiones aleatorias	—	159/162	✓	0.283322	215/216	✓
Serial	0.588596	40/40	✓	0.577037	40/40	✓
De complejidad lineal	0.122325	20/20	✓	0.090936	20/20	✓

Tabla 5.1: Resultado de las pruebas estadísticas del DRBG basado en cifrados por bloque (AES) para los niveles de seguridad de 112 y 128.

Pruebas estadísticas para el DRBG basado en cifrados por bloque (AES)						
Nivel de seguridad	192		256			
Prueba	Valor P	Proporción		Valor P	Proporción	
De frecuencia	0.964295	20/20	✓	0.162606	20/20	✓
De frecuencia en un bloque	0.739918	20/20	✓	0.162606	20/20	✓
De sumas acumulativas	0.637119	40/40	✓	0.325292	40/40	✓
De carreras	0.834308	19/20	✓	0.534146	20/20	✓
De la carrera más larga en un bloque	0.534146	19/20	✓	0.739918	20/20	✓
Del rango de matriz binaria	0.350485	20/20	✓	0.122325	20/20	✓
Espectral	0.637119	20/20	✓	0.637119	19/20	✓
De coincidencia sin superposición	0.496882	2925/2960	✓	0.495414	2923/2960	✓
De coincidencia con superposición	0.122325	20/20	✓	0.437274	19/20	✓
Estadística universal de Maurer	0.350485	20/20	✓	0.437274	20/20	✓
De entropía aproximada	0.834308	20/20	✓	0.534146	20/20	✓
De excusiones aleatorias	0.413551	87/88	✓	0.393867	88/88	✓
Variante de excusiones aleatorias	0.416464	198/198	✓	0.399044	197/198	✓
Serial	0.404927	39/40	✓	0.506506	40/40	✓
De complejidad lineal	0.350485	20/20	✓	0.534146	20/20	✓

Tabla 5.2: Resultado de las pruebas estadísticas del DRBG basado en cifrados por bloque (AES) para los niveles de seguridad de 192 y 256.

Pruebas estadísticas para el DRBG basado en funciones hash (SHA256)						
Nivel de seguridad	112			128		
Prueba	Valor P	Proporción		Valor P	Proporción	
De frecuencia	0.437274	20/20	✓	0.739918	20/20	✓
De frecuencia en un bloque	0.637119	20/20	✓	0.637119	19/20	✓
De sumas acumulativas	0.328236	40/40	✓	0.264105	40/40	✓
De carreras	0.006196	19/20	✓	0.739918	20/20	✓
De la carrera más larga en un bloque	0.275709	20/20	✓	0.739918	20/20	✓
Del rango de matriz binaria	0.350485	20/20	✓	0.637119	19/20	✓
Espectral	0.350485	19/20	✓	0.008879	20/20	✓
De coincidencia sin superposición	0.521362	2921/2960	✗	0.480227	2937/2960	✓
De coincidencia con superposición	0.637119	20/20	✓	0.911413	19/20	✓
Estadística universal de Maurer	0.637119	20/20	✓	0.048716	20/20	✓
De entropía aproximada	0.275709	20/20	✓	0.437274	20/20	✓
De excursiones aleatorias	0.407729	102/104	✓	—	62/64	✓
Variante de excursiones aleatorias	0.376856	232/234	✓	—	142/144	✓
Serial	0.404927	40/40	✓	0.637032	40/40	✓
De complejidad lineal	0.637119	20/20	✓	0.637119	20/20	✓

Tabla 5.3: Resultado de las pruebas estadísticas del DRBG basado en funciones hash (SHA256) para los niveles de seguridad de 112 y 128.

En la tabla 5.3 se observa que la única prueba que no pasó el DRBG basado en SHA256 con nivel de seguridad de 112, fue la de coincidencias sin superposición, situación que viene de no haberse aprobado más del 90 % de una de las subpruebas.

Pruebas estadísticas para el DRBG basado en funciones hash (SHA256)						
Nivel de seguridad	192			256		
Prueba	Valor P	Proporción		Valor P	Proporción	
De frecuencia	0.739918	20/20	✓	0.006196	20/20	✓
De frecuencia en un bloque	0.437274	20/20	✓	0.162606	20/20	✓
De sumas acumulativas	0.899301	40/40	✓	0.688519	40/40	✓
De carreras	0.911413	17/20	✗	0.350485	20/20	✓
De la carrera más larga en un bloque	0.834308	20/20	✓	0.911413	20/20	✓
Del rango de matriz binaria	0.911413	20/20	✓	0.122325	20/20	✓
Espectral	0.035174	20/20	✓	0.437274	19/20	✓
De coincidencia sin superposición	0.504889	2937/2960	✓	0.507982	2928/2960	✓

Continúa en siguiente página

<i>Continuación</i>					
Nivel de seguridad	192		256		
Prueba	Valor P	Proporción	Valor P	Proporción	
De coincidencia con superposición	0.739918	20/20	✓	0.834308	20/20
Estadística universal de Maurer	0.090936	20/20	✓	0.637119	20/20
De entropía aproximada	0.739918	20/20	✓	0.637119	20/20
De excursiones aleatorias	0.667135	78/80	✓	—	72/72
Variante de excursiones aleatorias	0.498556	174/180	✓	—	162/162
Serial	0.637032	39/40	✓	0.534146	38/40
De complejidad lineal	0.911413	20/20	✓	0.350485	19/20

Tabla 5.4: Resultado de las pruebas estadísticas del DRBG basado en funciones hash (SHA256) para los niveles de seguridad de 192 y 256.

En la tabla 5.4 se observa que el DRBG basado en SHA256 con nivel de seguridad de 192, no aprobó de forma general la prueba de carreras, esto por no llegar a las 18 ejecuciones aprobadas de 20.

Pruebas estadísticas para el DRBG basado en funciones hash (SHA512)					
Nivel de seguridad	112		128		
Prueba	Valor P	Proporción	Valor P	Proporción	
De frecuencia	0.213309	20/20	✓	0.637119	20/20
De frecuencia en un bloque	0.834308	20/20	✓	0.437274	20/20
De sumas acumulativas	0.911413	40/40	✓	0.637032	40/40
De carreras	0.534146	19/20	✓	0.739918	20/20
De la carrera más larga en un bloque	0.911413	20/20	✓	0.122325	19/20
Del rango de matriz binaria	0.834308	20/20	✓	0.834308	19/20
Espectral	0.637119	20/20	✓	0.739918	19/20
De coincidencia sin superposición	0.512787	2934/2960	✓	0.475790	2937/2960
De coincidencia con superposición	0.437274	20/20	✓	0.275709	20/20
Estadística universal de Maurer	0.739918	20/20	✓	0.275709	20/20
De entropía aproximada	0.834308	20/20	✓	0.739918	20/20
De excursiones aleatorias	—	72/72	✓	0.442340	77/80
Variante de excursiones aleatorias	—	161/162	✓	0.523098	180/180
Serial	0.493802	40/40	✓	0.585633	40/40
De complejidad lineal	0.534146	20/20	✓	0.350485	20/20

Continúa en siguiente página

<i>Continuación</i>					
Nivel de seguridad	112		128		
Prueba	Valor P	Proporción		Valor P	Proporción

Tabla 5.5: Resultado de las pruebas estadísticas del DRBG basado en funciones hash (SHA512) para los niveles de seguridad de 112 y 128.

Pruebas estadísticas para el DRBG basado en funciones hash (SHA512)						
Nivel de seguridad	192		256			
Prueba	Valor P	Proporción		Valor P	Proporción	
De frecuencia	0.162606	20/20	✓	0.534146	20/20	✓
De frecuencia en un bloque	0.739918	20/20	✓	0.739918	19/20	✓
De sumas acumulativas	0.336147	40/40	✓	0.637032	40/40	✓
De carreras	0.534146	20/20	✓	0.637119	20/20	✓
De la carrera más larga en un bloque	0.739918	20/20	✓	0.834308	20/20	✓
Del rango de matriz binaria	0.534146	20/20	✓	0.739918	20/20	✓
Espectral	0.066882	19/20	✓	0.275709	19/20	✓
De coincidencia sin superposición	0.489630	2934/2960	✓	0.484244	2929/2960	✓
De coincidencia con superposición	0.350485	20/20	✓	0.437274	20/20	✓
Estadística universal de Maurer	0.834308	20/20	✓	0.739918	19/20	✓
De entropía aproximada	0.275709	20/20	✓	0.964295	20/20	✓
De excusiones aleatorias	0.503591	104/104	✓	0.481113	78/80	✓
Variante de excusiones aleatorias	0.235885	234/234	✓	0.572188	178/180	✓
Serial	0.688519	39/40	✓	0.555009	37/40	✓
De complejidad lineal	0.637119	19/20	✓	0.739918	20/20	✓

Tabla 5.6: Resultado de las pruebas estadísticas del DRBG basado en funciones hash (SHA512) para los niveles de seguridad de 192 y 256.

Con base en los resultados de las pruebas estadísticas, se puede decir que los DRBG implementados son aleatorios. Aunque hay que tomar en cuenta que el número de ejecuciones que se realizaron para las pruebas (20) es aun un número pequeño, pero hacer un mayor número de pruebas consumiría mas recursos de los que se tienen.

	Tokenización	Detokenización
BPS	359.088 μ s	357.620 μ s
FFX	274.905 μ s	275.147 μ s
TKR	74.424 ms	669.203 μ s
AHR	70.041 ms	719.729 μ s
DRBG	75.247 ms	696.643 μ s

Tabla 5.7: Comparación de tiempos de tokenización.

5.3.2. Comparación de desempeño

Como se indicó en la introducción de este trabajo, la tokenización es un proceso relativamente nuevo y la mayoría de los proveedores de este servicio aprovechan la confusión y desinformación existente para atraer a clientes potenciales, pues pocos comparten la manera en que crean sus TOKENS. Esta sección busca resarcir un poco el daño, presentando una comparación entre los tiempos de respuesta de tokenización y detokenización de los algoritmos implementados.

En la tabla 5.7 se puede observar el tiempo promedio que le toma a cada algoritmo hacer una operación de tokenización y detokenización; estos valores se obtuvieron al ejecutar varias veces 15K operaciones de tokenización y detokenización, luego obtener el promedio y, finalmente, dividir entre 15K. Los algoritmos están divididos en reversibles y no reversibles, pues, dadas sus características, no sería muy parejo compararlos entre sí; por ejemplo, basta hacer una consulta en la base de datos para hacer la detokenización en los algoritmos irreversibles; mientras que en los reversibles hay que hacer el proceso inverso, o uno equivalente, para poder obtener de nuevo el PERSONAL ACCOUNT NUMBER (PAN).

Los datos de la tabla 5.7 fueron utilizados para crear una serie de gráficas: en la figura 5.1 se comparan los tiempos de tokenización y detokenización de todos los algoritmos, tanto reversibles como irreversibles; la velocidad de los algoritmos reversibles es notoria, en especial al momento de la operación de tokenización. Posteriormente, en la figura 5.2, se compararon solo los algoritmos reversibles (BPS y FFX), siendo el segundo el más rápido de los dos; la similitud del comportamiento de estos algoritmos en ambas operaciones se debe a que se realizan casi las mismas operaciones para cifrar como para decifrar. Finalmente, se compararon los tiempos para los algoritmos irreversibles en las gráficas de la figura 5.3: durante las operaciones de tokenización, aunque están relativamente cerrados los tiempos, AHR queda como el más rápido de los tres, mientras que DRBG queda en segundo lugar; es menester recordar porqué estos algoritmos toman más tiempo: AHR utiliza CIFRADO DE CAMINATA CÍCLICA, y, al generar un TOKEN, consulta la base de datos, pues permite tener más de un TOKEN por PAN; además, tanto TKR2 como DRBG necesitan bits pseudoaleatorios, y si el generador de bits pseudoaleatorios tarda en obtenerlos, los algoritmos se ven afectados; finalmente, en las operaciones de detokenización, DRBG es el más rápido de todos, mientras que AHR queda como el más lento de los tres. En la figura 5.4 se puede ver la diferencia

de tiempos entre las operaciones de tokenización y detokenización; los tiempos de los algoritmos reversibles son casi idénticos en ambas operaciones, mientras que se observa un cambio notorio entre los tiempos de tokenización y detokenización de los irreversibles.

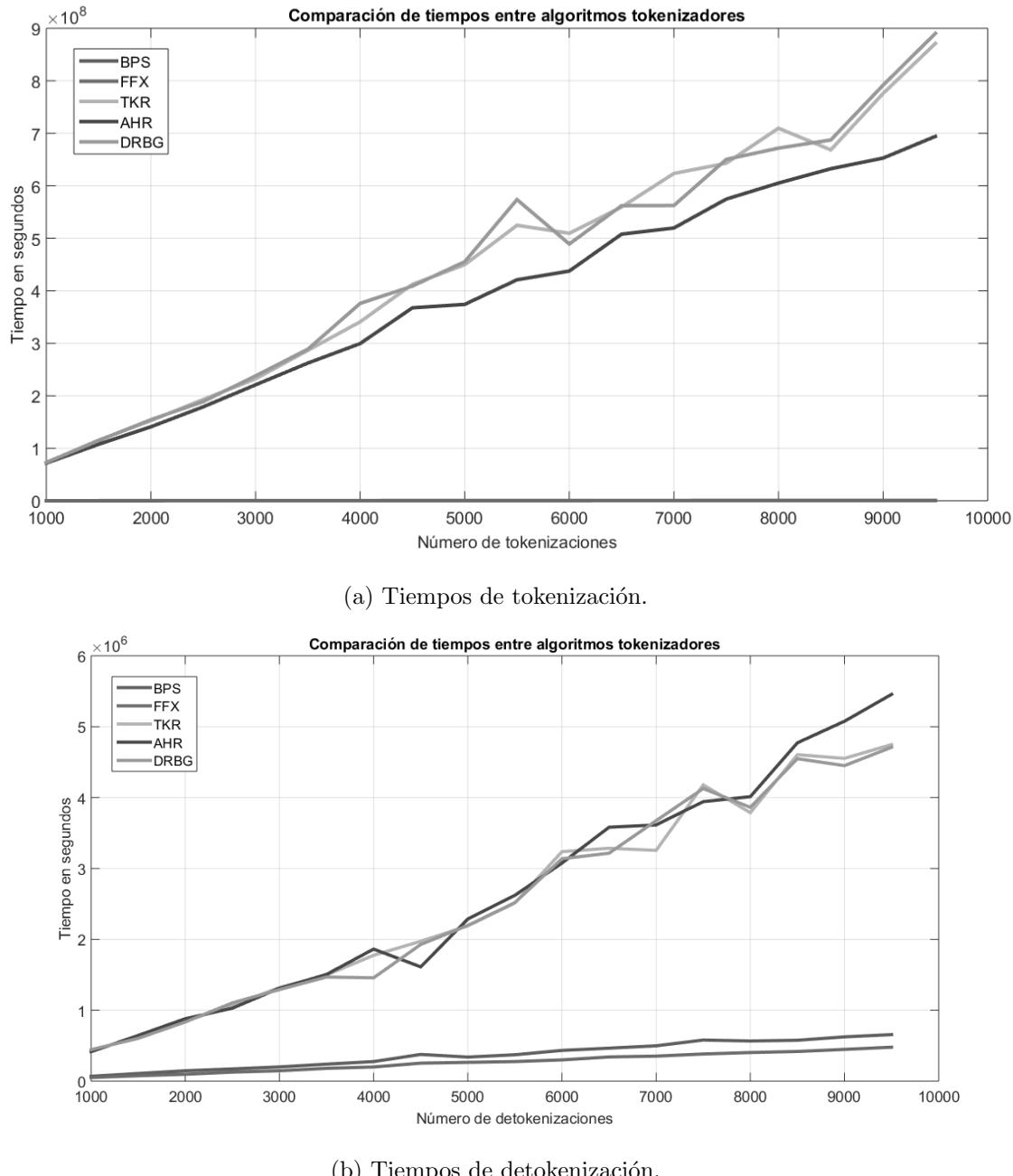
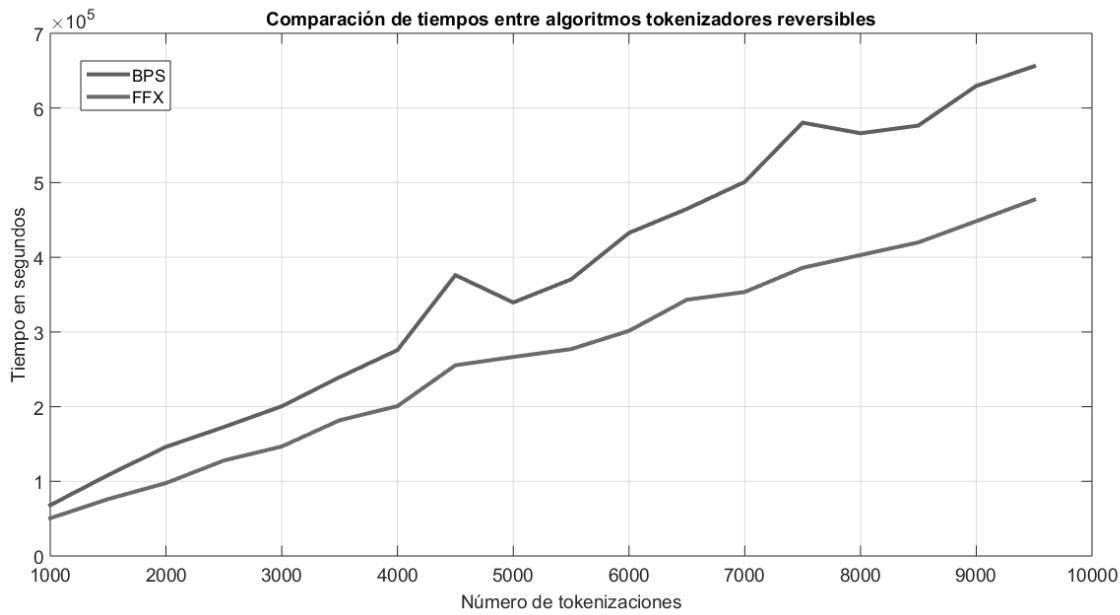
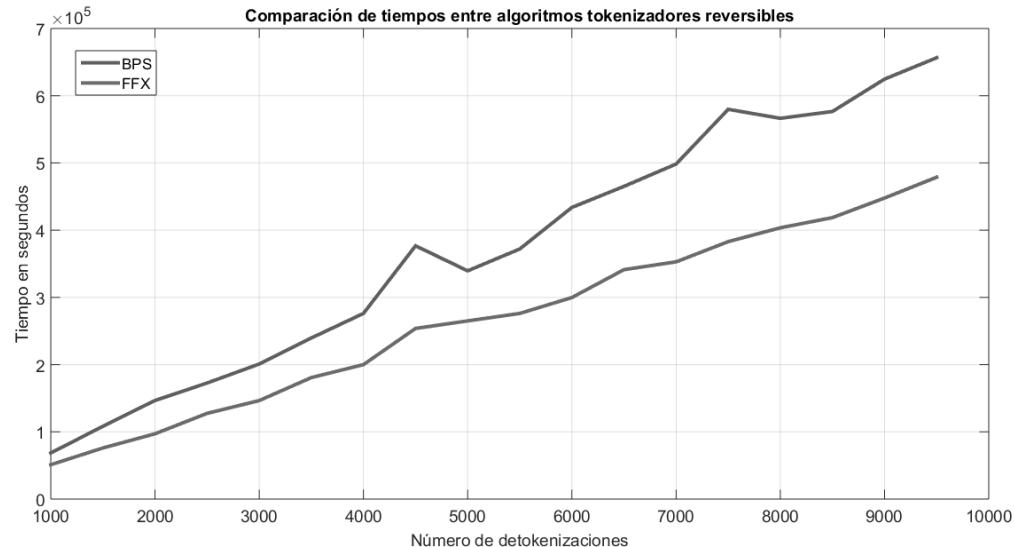


Figura 5.1: Comparación de tiempos entre algoritmos tokenizadores.

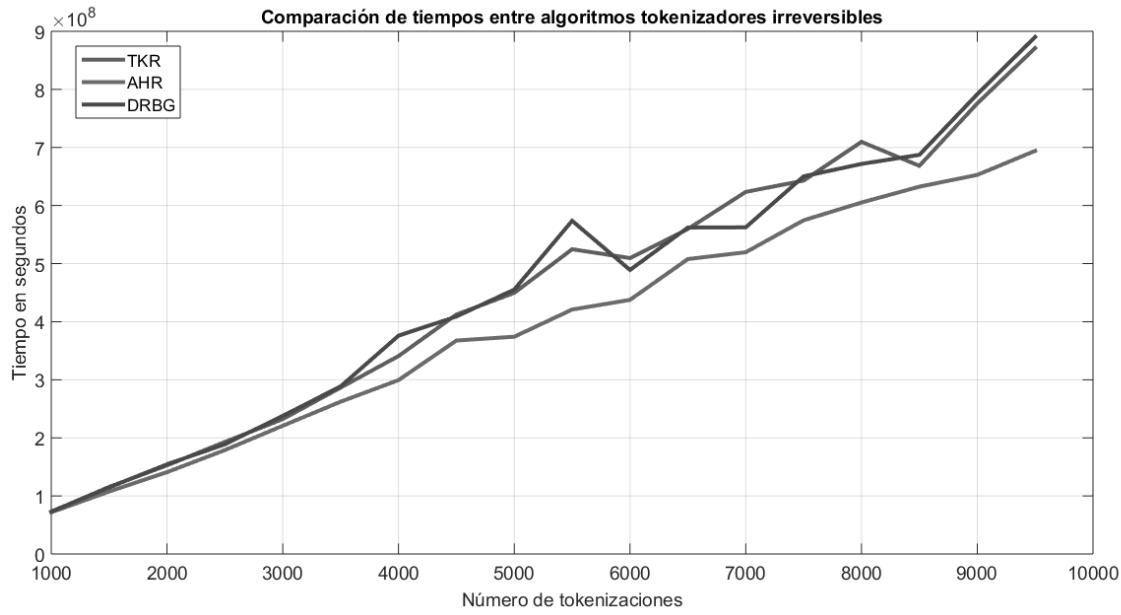


(a) Tiempos de tokenización.

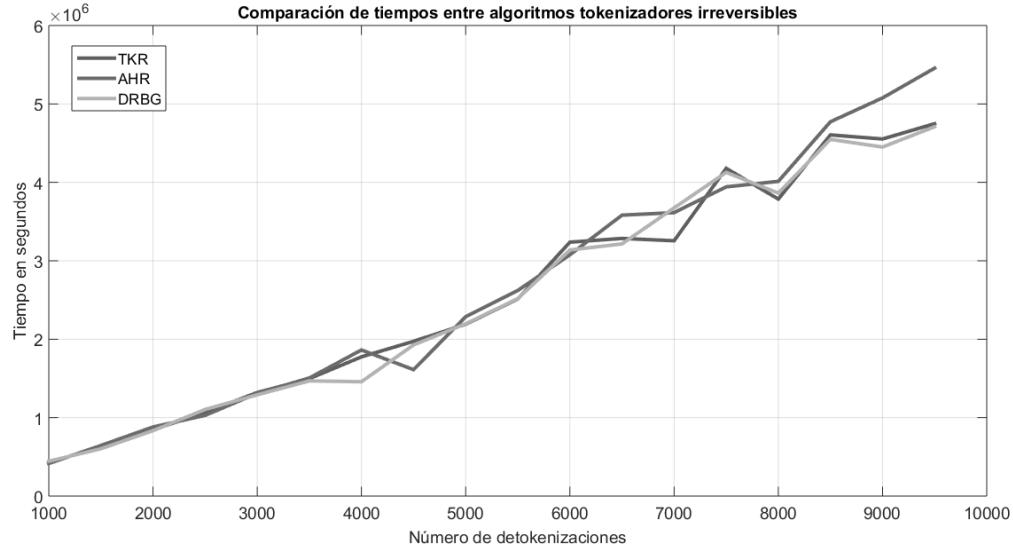


(b) Tiempos de detokenización.

Figura 5.2: Comparación de tiempos entre algoritmos tokenizadores reversibles.



(a) Tiempos de tokenización.



(b) Tiempos de detokenización.

Figura 5.3: Comparación de tiempos entre algoritmos tokenizadores irreversibles.

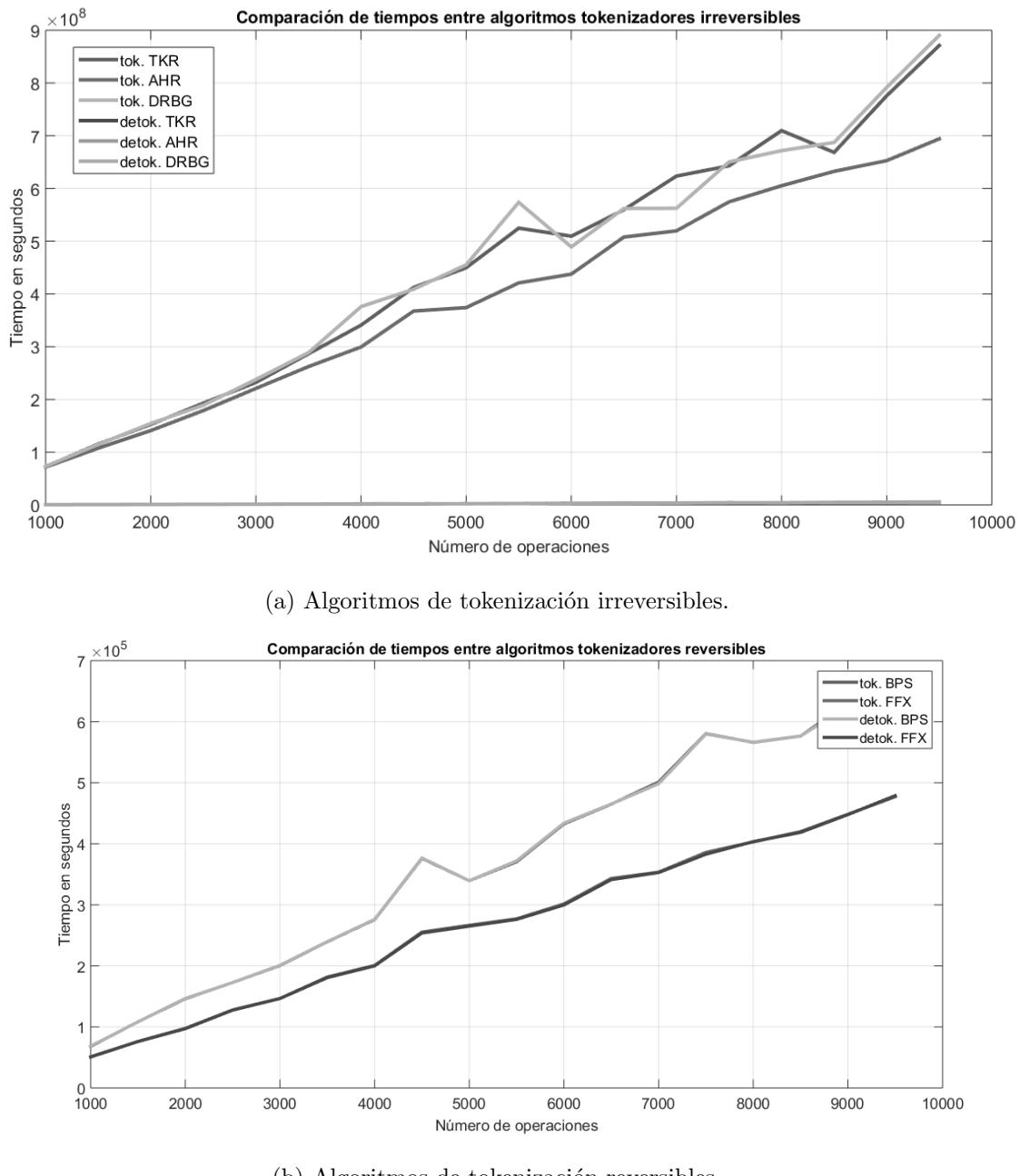


Figura 5.4: Comparación de tiempos de tokenización y detokenización.

Parte II

Aplicación web de sistema tokenizador

«*People who think they know everything
are a great annoyance to those of us who
do.*»

ISAAC ASIMOV.

Capítulo 6

Análisis y diseño de aplicación web

«We live in a society exquisitely dependent on science and technology, in which hardly anyone knows anything about science and technology.»

Cosmos, CARL SAGAN.

En este capítulo se abordan los temas referentes al análisis y al diseño del segundo módulo del proyecto: la interfaz web al sistema tokenizador.

6.1. Análisis

A lo largo de esta sección se utilizan de forma equivalente los términos *sistema*, *aplicación*, *aplicación web* y *programa*. Todos hacen referencia al segundo prototipo del proyecto. Los términos *pantalla*, *interfaz*, *interfaz gráfica*, *página*, *ventana* y *formulario* (los dos últimos solamente en donde aplique la denominación) son utilizados por igual para referirse al estado gráfico del navegador en un momento predeterminado.

En esta sección se utilizarán **negritas** para referirse a estados (**verificado**, **aceptado**) o tipos de un elemento (**cliente**, **administrador**) e *itálicas* para referirse a elementos de las interfaces.

El desarrollo (y por consiguiente, la lectura) de las secciones de reglas de negocio, requerimientos, casos de uso, interfaces de usuario y mensajes no fue hecho de forma secuencial, dado el grado de paralelismo que hay entre estas secciones. Quizá la forma más sencilla de leerlo sea pasar directamente a los casos de uso, y de ahí ir buscando las referencias a las demás secciones.

6.1.1. Reglas de negocio

El formato de las EXPRESIONES REGULARES, por cuestiones de optimización en tiempos de desarrollo, se presenta según el formato de *JavaScript*, definido en [51].

RNAPI-01 Composición de números de tarjeta.

Un número de tarjeta queda definido por la expresión regular:

`/^ [0-9] {12, 19} $/`

El número se divide en tres partes: el identificador del emisor (primeros 6 dígitos), el número de cuenta y el dígito verificador (último dígito). Para más detalles sobre este tema consultar la sección 2.5. El dígito verificador se obtiene de tal forma que el algoritmo de Luhn (sección 2.5.3) dé como resultado cero.

RNAPI-02 Composición de TOKEN.

El formato de un token es el mismo que para un número de tarjeta (RNAPI-01 COMPOSICIÓN DE NÚMEROS DE TARJETA) con la única excepción de que el último dígito, el dígito verificador, se calcule de forma que el algoritmo de Luhn (sección 2.5.3) dé como resultado un uno.

RNAPI-03 Unicidad del correo electrónico.

El correo electrónico con el que se registra a los usuarios **cliente** y **administrador** debe ser único.

RNAPI-04 Formato de correos electrónicos.

Un correo electrónico queda definido, de acuerdo al REQUEST FOR COMMENTS (RFC) 5322 [52], por la expresión regular:

```
/^(([^\<>()\[\]\\.,;:\s@"]+(\.\[^\<>()\[\]\\.,;:\s@"]+)*|(.+))@  
((\[[0-9]{1,3}\.\[0-9]{1,3}\.\[0-9]{1,3}\.\[0-9]{1,3}\])|(([a-zA-Z\-\-0-9]+\.)+  
[a-zA-Z]{2,}))$/
```

RNAPI-05 Formato de contraseñas.

El formato de una contraseña está regido por la expresión regular:

```
/^[\x20-\x7F]{8,24}$/
```

Esto es, entre 8 y 24 caracteres AMERICAN STANDARD CODE FOR INFORMATION INTERCHANGE (ASCII) imprimibles.

RNAPI-06 Tipos de algoritmos tokenizadores.

Existen dos tipos de algoritmos tokenizadores:

- Reversibles
- Irreversibles

RNAPI-07 Tipos de usuario.

Existen cuatro tipos de usuarios:

- Visitante
- Cliente
- Administrador
- Aplicación cliente

Al abrir la aplicación, todos los usuarios son visitantes; pues los clientes y los administradores deben iniciar sesión para ser identificados por el sistema; un visitante puede convertirse en cliente al registrarse y ser aprobado. Las transiciones entre los tipos de usuarios pueden observarse en el diagrama 6.1. Una aplicación cliente es aquella que utiliza el servicio de tokenización (por ejemplo, una tienda en línea); cada cliente representa a una aplicación cliente.

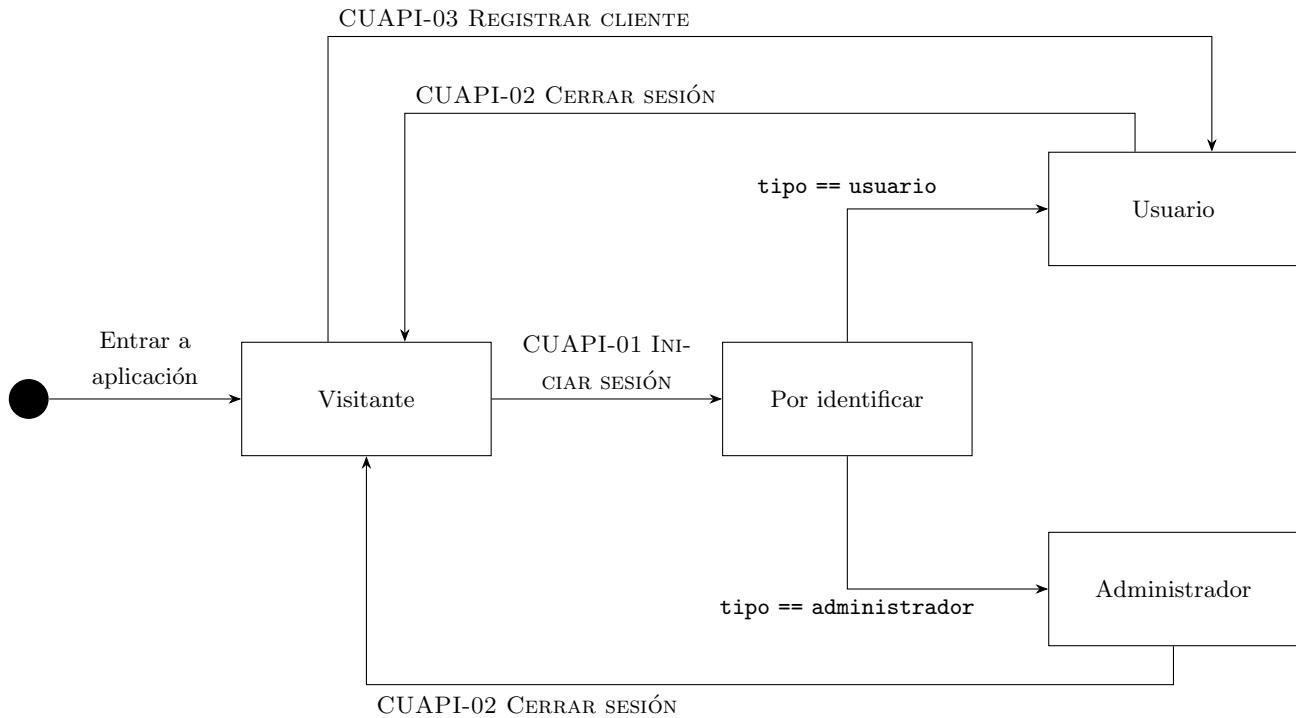


Figura 6.1: Diagrama de estados de actores.

RNAPI-08 Estados de un cliente.

Un cliente puede tener los siguientes estados:

- En espera
- Aprobado
- Rechazado
- En cambio de llave
- En lista negra

Cuando un cliente se registra, se encuentra en estado **en espera**, con correo **no verificado**; cuando verifica su dirección de correo electrónico, pasa a **en espera** con correo electrónico **verificado**; una vez que el administrador lo aprueba, pasa a **aprobado**; estando en estado **en espera** con correo **verificado**, un administrador puede pasar a un cliente al estado **rechazado**. Cuando inicia un refresco de llaves, pasa a **en cambio de llaves**. Finalmente, el administrador puede cambiar su estado a **en lista negra** (y regresarlo a **aprobado** o **en cambio de llaves**). Estas transiciones se pueden observar en la Figura 6.2.

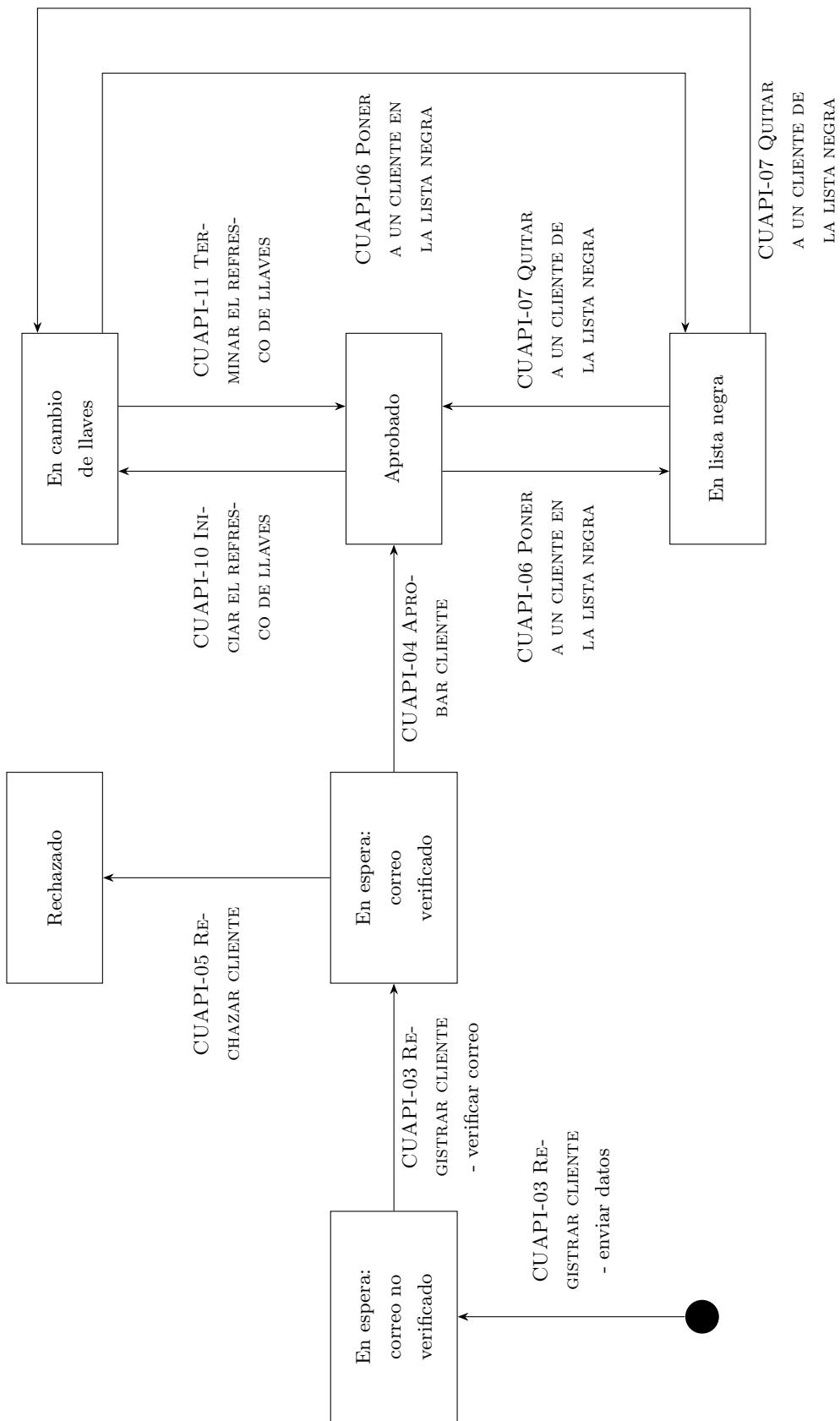


Figura 6.2: Diagrama de estados de un cliente.

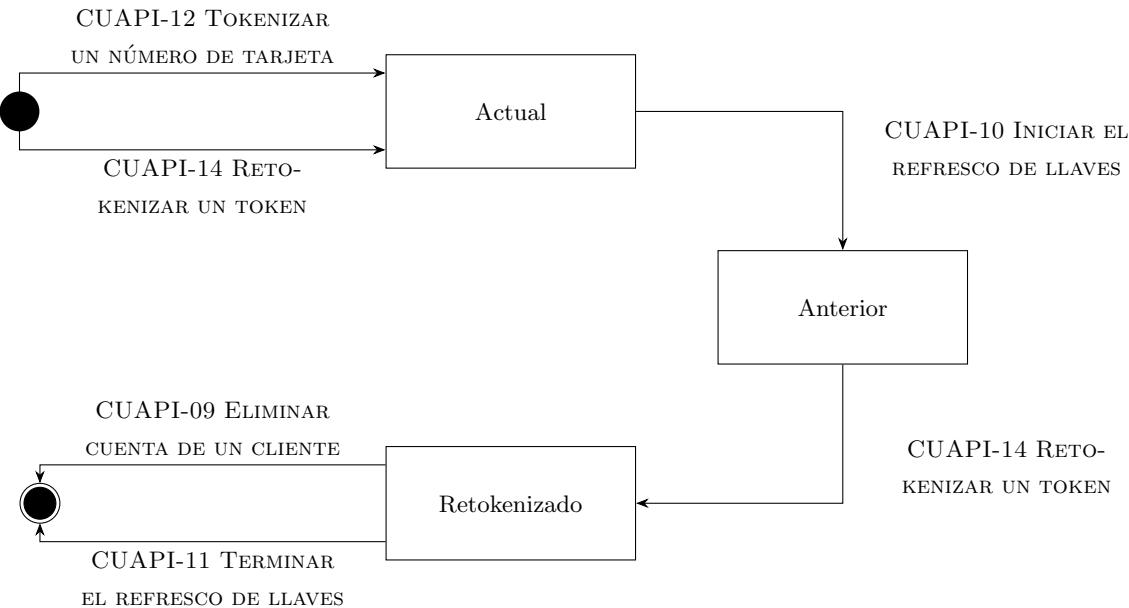


Figura 6.3: Diagrama de estados de tokens.

RNAPI-09 Estados de un TOKEN.

Un token puede tener los siguientes estados:

- Actual
- Anterior
- Retokenizado

Cuando un token es creado, su estado es **actual**. Cuando un cliente inicia el refresco de llaves, todos sus tokens cambian de estado a **anterior**. Cuando un cliente solicita la retokenización de un token, se cambia su estado a **retokenizado**. Al terminar el refresco de llaves o al eliminar la cuenta de un cliente, el token se elimina. Estas transiciones se pueden observar en la Figura 6.3.

RNAPI-10 Estados de una llave.

Una llave puede tener los siguientes estados:

- Actual
- Anterior

Cuando una llave es creada, su estado es **actual**; cuando el cliente inicia el refresco de llaves, la llave existente cambia a estado **anterior**. La llave se elimina cuando termina el refresco de llaves o cuando se elimina la cuenta de un cliente. Estas transiciones se pueden observar mejor en la Figura 6.4.

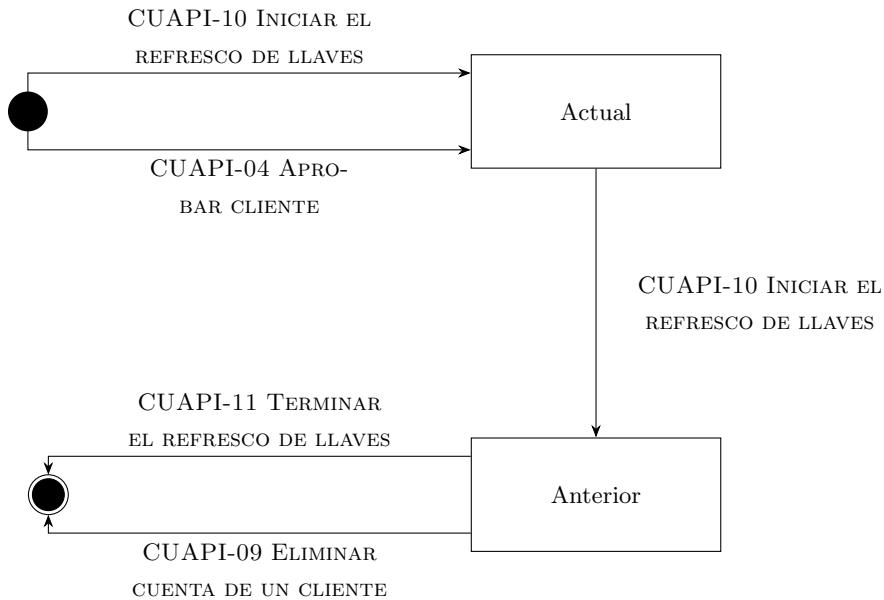


Figura 6.4: Diagrama de estados de llaves.

RNAPI-11 Estados de un correo electrónico.

Un correo electrónico puede tener los siguientes estados:

- Verificado
- No verificado

Cuando un correo es registrado o actualizado , su estado es **no verificado**; cambia su estado a **verificado** cuando el cliente presiona un vínculo de validación válido (véase RNAPI-21 EXPIRACIÓN DE VÍNCULOS DE VERIFICACIÓN DE REGISTRO); se elimina cuando se borra la cuenta del cliente o cuando este actualiza su información. Estas transiciones se pueden observar mejor en la Figura 6.5.

RNAPI-12 Contador de malas acciones.

Todo cliente tiene asociado un cliente de malas acciones que aumenta de la siguiente manera:

- Aumentar en tres cuando un cliente intenta detokenizar tokens válidos que no son suyos.
- Aumentar en uno cuando un cliente intenta detokenizar tokens no válidos.
- Aumentar en uno cuando un cliente intente acceder a páginas que no tiene acceso.

Cuando un cliente es creado, el contador está en cero.

RNAPI-13 Sancionar cliente.

Cuando el contador de malas acciones de un cliente sea mayor o igual a 10, el estado del cliente deberá ser actualizado a **en lista negra**.

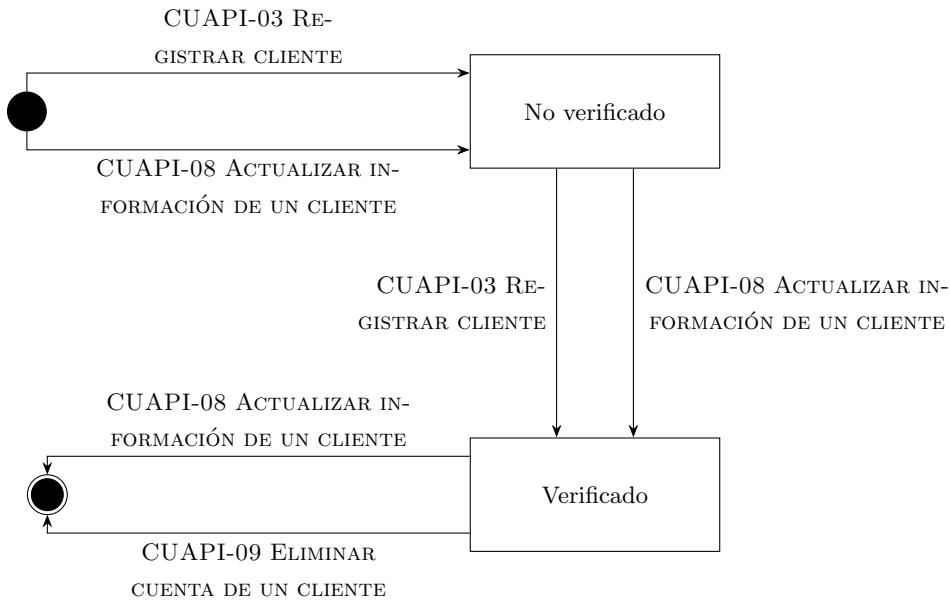


Figura 6.5: Diagrama de estados de un correo electrónico.

RNAPI-14 Petición de tokenización.

Cuando un cliente solicite una operación de tokenización, se debe utilizar la llave con estado **actual** para obtener el token.

RNAPI-15 Petición de detokenización.

Cuando un cliente en estado **aceptado** solicite una operación de detokenización, se debe utilizar la llave con estado **actual** para obtener el número de tarjeta.

RNAPI-16 Petición de detokenización en proceso de refresco.

Cuando un cliente solicite una operación de detokenización y se encuentre en estado **en cambio de llaves**, se debe especificar en la petición si se desea utilizar la llave nueva o anterior. En caso de no ser especificada, se realizará la operación utilizando la llave con estado **actual**.

RNAPI-17 Habilitación de retokenización.

La operación de retokenización solo está habilitada para los clientes cuyo estado es **en cambio de llaves**.

RNAPI-18 Criptoperiodo de una llave.

El criptoperiodo de todas las llaves usadas para la tokenización (y detokenización) es de seis meses (ver el Apéndice E).

RNAPI-19 Unicidad de número de tarjeta por cliente.

Un cliente solo puede tener un token por tarjeta; esto es, no puede realizar, para un mismo número de tarjeta, la operación de tokenización más de una vez.

RNAPI-20 Hipervínculo de verificación.

Un vínculo de confirmación está conformado por dos partes; la dirección base del sitio y un valor hash obtenido mediante el correo y la contraseña del usuario que se va a verificar.

RNAPI-21 Expiración de vínculos de verificación de registro.

El vínculo para verificar una cuenta es efectivo solamente durante 24 horas; después de este tiempo, el vínculo deja de ser válido y se elimina el registro del usuario.

RNAPI-22 Expiración de vínculos de verificación de actualización.

El vínculo para verificar la actualización de una cuenta es efectivo indefinidamente.

RNAPI-23 Aprobación de clientes.

Un administrador puede aprobar o rechazar usuarios tipo **cliente** cuando estos tengan estado **en espera** con correo **verificado** en un periodo de 24 HRS desde que el cliente verificó su correo.

6.1.2. Requerimientos

REQFAPI-01 Gestionar sesión.

Funcionales Una aplicación web debe permitir a los usuarios **cliente** y **administrador** iniciar y cerrar sesión. Deben proporcionar los siguientes datos para iniciar sesión:

- Correo electrónico
- Contraseña

REQFAPI-02 Validar credenciales al iniciar sesión.

La contraseña introducida debe coincidir con el correo ingresado.

REQFAPI-03 Validar estado de clientes al iniciar sesión.

Aquellos **clientes** que se encuentren en alguno de los siguientes estados no deben poder iniciar sesión: **en espera, rechazado o en lista negra.**

REQFAPI-04 Validar estado de correo al iniciar sesión.

El correo del cliente debe encontrarse en estado **válido**.

REQAPI-05 Registrar cliente.

La aplicación web debe permitir a los usuarios de tipo **visitante** registrarse; el visitante debe proporcionar los siguientes datos:

- Correo electrónico válido
- Contraseña
- Confirmación de contraseña

REQAPI-06 Verificar cliente.

La aplicación web debe permitir a un usuario de tipo **cliente** verificar su correo mediante un enlace enviado al registrarse.

REQAPI-07 Verificar nuevo correo.

La aplicación web debe permitir a un usuario de tipo **cliente** verificar su nuevo correo mediante un enlace enviado al terminar de editar sus datos.

REQAPI-08 Aprobar cliente.

La aplicación web debe permitir a un usuario de tipo **administrador** aprobar y rechazar usuarios de tipo **cliente**.

REQAPI-09 Mostrar clientes en espera.

La aplicación web debe permitir a un usuario de tipo **administrador** consultar la lista de **clientes** en estado **en espera** con correo **verificado**.

REQAPI-10 Mostrar clientes aprobados.

La aplicación web debe permitir a un usuario de tipo **administrador** consultar la lista de **clientes** en estado **aprobado**.

REQAPI-11 Mostrar clientes en lista negra.

La aplicación web debe permitir a un usuario de tipo **administrador** consultar la lista de **clientes** en estado **en lista negra**.

REQAPI-12 Administrar lista negra.

La aplicación web debe permitir a un usuario de tipo **administrador** poner y quitar a un **cliente** de la lista negra.

REQAPI-13 Quitar cliente de lista negra.

La aplicación web debe permitir a un usuario de tipo **administrador** quitar a un **cliente** de la lista negra.

REQFAPI-14 Actualizar datos de cliente.

La aplicación web debe permitir a un usuario de tipo **cliente** actualizar sus datos (correo electrónico y contraseña).

REQFAPI-15 Eliminar datos de un cliente.

La aplicación web debe permitir a un usuario de tipo **cliente** eliminar su cuenta y datos del sistema.

REQFAPI-16 Administrar refresco de llaves.

La aplicación web debe permitir, a un **cliente**, iniciar o terminar el refresco de sus llaves.

REQFAPI-17 Recordatorio de refresco de llaves.

El sistema debe mostrar un aviso en la página de inicio de un **cliente** y enviar una notificación diaria a los clientes que tengan llaves cuyo criptoperiodo haya caducado.

REQFAPI-18 Tokenizar número de tarjeta.

La aplicación web debe permitir, a una **aplicación cliente**, realizar la tokenización de un número de tarjeta. La aplicación cliente debe proporcionar los siguientes datos:

- Número de tarjeta a tokenizar.
- Algoritmo tokenizador.
- Credenciales.

La operación regresa un token.

SUBREQFAPI-18/1 Validar estado de clientes al tokenizar.

Para realizar una operación de tokenización, el **cliente** asociado a la **aplicación cliente** debe encontrarse en estado **aceptado** o **en cambio de llaves**.

REQFAPI-19 Detokenizar un TOKEN.

La aplicación web debe permitir, a una **aplicación cliente**, realizar la detokenización de un token.

La aplicación cliente debe proporcionar los siguientes datos:

- Token.
- Algoritmo tokenizador.
- Credenciales.
- [Opcional] Versión de llaves a utilizar.

La operación regresa un número de tarjeta.

SUBREQFAPI-19/1 Validar estado de clientes al detokenizar.

Para realizar una operación de detokenización, el **cliente** asociado a la **aplicación cliente** debe encontrarse en estado **aceptado** o **en cambio de llaves**.

REQFAPI-20 Retokenizar un TOKEN.

La aplicación web debe permitir, a una **aplicación cliente** cuyo **cliente** ha iniciado el refresco de llaves, realizar la retokenización de un token. La aplicación cliente debe proporcionar los siguientes datos:

- Token.
- Algoritmo tokenizador.
- Credenciales.

La operación regresa el token actualizado.

SUBREQFAPI-20/1 Validar estado de clientes al retokenizar.

Para realizar una operación de retokenización, el **cliente** asociado a la **aplicación cliente** debe encontrarse en estado **en cambio de llaves**.

REQFAPI-21 Algoritmos para tokenizar.

Las operaciones de tokenización, detokenización y retokenización (especificadas en REQFAPI-18 TOKENIZAR NÚMERO DE TARJETA, REQFAPI-19 DETOKENIZAR UN TOKEN y REQFAPI-20 RETOKENIZAR UN TOKEN respectivamente) deben especificar el algoritmo tokenizador con el que se realizará la operación. Los algoritmos disponibles son:

- BPS
- FFX
- TKR
- AHR
- DRBG

REQFAPI-22 Mostrar promoción del servicio tokenizador.

La aplicación web debe tener una página que promocione el servicio de tokenización, esta página también debe explicar qué es la tokenización, qué es un servicio de tokenización y cómo funcionan los algoritmos implementados.

REQFAPI-23 Mostrar documentación del servicio tokenizador.

La aplicación web debe tener una página donde se muestre la documentación de la API.

REQFAPI-24 Notificar cliente.

La aplicación web debe notificar a un **cliente** siempre que le envíe un correo de verificación.

REQFAPI-25 Notificar cliente por correo electrónico.

La aplicación web debe notificar por correo electrónico a un **cliente** siempre que un **administrador** cambie su estado.

REQAPI-26 Confirmación de un cliente.

La aplicación web debe pedir la confirmación del **cliente** siempre que quiera realizar una de las siguientes acciones:

- Eliminar su cuenta.
- Terminar refresco de llaves con tokens en estado **anterior**.

REQAPI-27 Confirmación de un administrador.

La aplicación web debe pedir la confirmación del **administrador** siempre que quiera realizar una de las siguientes acciones:

- Rechazar a un cliente.
- Vetar a un cliente.
- Terminar refresco de llaves con tokens en estado **anterior**.

REQAPI-28 Cambiar estado de un cliente.

La aplicación web debe permitir cambiar el estado de un **cliente**:

- **en espera**
- **aprobado**
- **rechazado**
- **en cambio de llaves**
- **en lista negra**

REQAPI-29 Cambiar estado de un token.

La aplicación web debe permitir cambiar el estado de un token:

- **actual**
- **anterior**
- **retokenizado**

REQAPI-30 Cambiar estado de una llave.

La aplicación web debe permitir cambiar el estado de una llave:

- **actual**
- **anterior**

REQFAPI-31 Cambiar estado de un correo.

La aplicación web debe permitir cambiar el estado de un correo:

- **verificado**
- **no verificado**

No funcionales

REQNFAPI-01 Sobre los tamaños de pantallas.

Las interfaces gráficas de la aplicación deben estar diseñadas para responder a distintos tamaños de pantalla. En particular, se debe mostrar cómo se ve la interfaz para los siguientes tamaños (las unidades están en pixeles): muy grande (1920x1080), grande (1280x800), mediano (900x960), pequeño (600x960) y muy pequeño (480x800).

REQNFAPI-02 Sobre el almacenamiento de contraseñas.

Las contraseñas nunca se almacenan en la base de datos. Lo que se almacena es un hash (sección 2.3); al momento de identificar a un usuario se calcula el hash de la contraseña introducida y se compara contra el que está almacenado en la base de datos. El algoritmo utilizado para estos hash debe de ser SECURE HASH ALGORITHM (SHA)-256.

REQNFAPI-03 Sobre los encabezados de autenticación.

Todas las peticiones de tokenización, detokenización y retokenización deben hacer uso de los encabezados de autenticación provistos por el protocolo HTTPS.

6.1.3. Casos de uso

A continuación se detallan los casos de uso del sistema. En el diagrama 6.6 se muestra una vista general de todos junto con su relación con los actores (regla de negocio RNAPI-07 TIPOS DE USUARIO) del sistema; mientras que en la tabla 6.1 se muestra la relación entre casos de uso y requerimientos.

Requerimiento	Caso de uso
REQFAPI-01 GESTIONAR SESIÓN	
REQFAPI-02 VALIDAR CREDENCIALES AL INICIAR SESIÓN	
REQFAPI-03 VALIDAR ESTADO DE CLIENTES AL INICIAR SESIÓN	
REQFAPI-04 VALIDAR ESTADO DE CORREO AL INICIAR SESIÓN	
REQFAPI-09 MOSTRAR CLIENTES EN ESPERA	CUAPI-01 INICIAR SESIÓN
REQFAPI-10 MOSTRAR CLIENTES APROBADOS	
REQFAPI-11 MOSTRAR CLIENTES EN LISTA NEGRA	
REQFAPI-17 RECORDATORIO DE REFRESCO DE LLAVES	
REQFAPI-01 GESTIONAR SESIÓN	CUAPI-02 CERRAR SESIÓN
REQFAPI-05 REGISTRAR CLIENTE	
REQFAPI-06 VERIFICAR CLIENTE	
REQFAPI-28 CAMBIAR ESTADO DE UN CLIENTE	CUAPI-03 REGISTRAR CLIENTE
REQFAPI-31 CAMBIAR ESTADO DE UN CORREO	
REQFAPI-08 APROBAR CLIENTE	
REQFAPI-09 MOSTRAR CLIENTES EN ESPERA	
REQFAPI-25 NOTIFICAR CLIENTE POR CORREO ELECTRÓNICO	CUAPI-04 APROBAR CLIENTE
REQFAPI-28 CAMBIAR ESTADO DE UN CLIENTE	
REQFAPI-30 CAMBIAR ESTADO DE UNA LLAVE	
REQFAPI-08 APROBAR CLIENTE	
REQFAPI-09 MOSTRAR CLIENTES EN ESPERA	
REQFAPI-25 NOTIFICAR CLIENTE POR CORREO ELECTRÓNICO	CUAPI-05 RECHAZAR CLIENTE
REQFAPI-27 CONFIRMACIÓN DE UN ADMINISTRADOR	
REQFAPI-28 CAMBIAR ESTADO DE UN CLIENTE	
REQFAPI-10 MOSTRAR CLIENTES APROBADOS	
REQFAPI-12 ADMINISTRAR LISTA NEGRA	
REQFAPI-25 NOTIFICAR CLIENTE POR CORREO ELECTRÓNICO	CUAPI-06 PONER A UN CLIENTE EN LA LISTA NEGRA
REQFAPI-27 CONFIRMACIÓN DE UN ADMINISTRADOR	
REQFAPI-28 CAMBIAR ESTADO DE UN CLIENTE	
REQFAPI-11 MOSTRAR CLIENTES EN LISTA NEGRA	
REQFAPI-12 ADMINISTRAR LISTA NEGRA	
REQFAPI-25 NOTIFICAR CLIENTE POR CORREO ELECTRÓNICO	CUAPI-07 QUITAR A UN CLIENTE DE LA LISTA NEGRA
REQFAPI-28 CAMBIAR ESTADO DE UN CLIENTE	
REQFAPI-14 ACTUALIZAR DATOS DE CLIENTE	
REQFAPI-24 NOTIFICAR CLIENTE	
REQFAPI-31 CAMBIAR ESTADO DE UN CORREO	CUAPI-08 ACTUALIZAR INFORMACIÓN DE UN CLIENTE
REQFAPI-15 ELIMINAR DATOS DE UN CLIENTE	CUAPI-09 ELIMINAR CUENTA DE UN CLIENTE

Continúa en siguiente página

<i>Continuación</i>	
Requerimiento	Caso de uso
REQFAPI-26 CONFIRMACIÓN DE UN CLIENTE	CUAPI-10 INICIAR EL REFresco DE LLAVES
REQFAPI-16 ADMINISTRAR REFresco DE LLAVES	
REQFAPI-29 CAMBIAR ESTADO DE UN TOKEN	
REQFAPI-30 CAMBIAR ESTADO DE UNA LLAVE	
REQFAPI-28 CAMBIAR ESTADO DE UN CLIENTE	CUAPI-11 TERMINAR EL REFresco DE LLAVES
REQFAPI-16 ADMINISTRAR REFresco DE LLAVES	
REQFAPI-28 CAMBIAR ESTADO DE UN CLIENTE	
REQFAPI-26 CONFIRMACIÓN DE UN CLIENTE	CUAPI-12 TOKENIZAR UN NÚMERO DE TARJETA
REQFAPI-18 TOKENIZAR NÚMERO DE TARJETA	
REQFAPI-21 ALGORITMOS PARA TOKENIZAR	CUAPI-13 DETOKENIZAR UN TOKEN
REQFAPI-19 DETOKENIZAR UN TOKEN	
REQFAPI-21 ALGORITMOS PARA TOKENIZAR	CUAPI-14 RETOKENIZAR UN TOKEN
REQFAPI-20 RETOKENIZAR UN TOKEN	
REQFAPI-21 ALGORITMOS PARA TOKENIZAR	
REQFAPI-29 CAMBIAR ESTADO DE UN TOKEN	

Tabla 6.1: Relación de casos de uso y requerimientos

CUAPI-01 Iniciar sesión.

Caso que permite que un **visitante** se identifique ante el sistema. El visitante introduce su correo electrónico y contraseña y, dependiendo de si es un **cliente** o un **administrador**, es redirigido a la página de inicio correspondiente.

Trayectoria principal.

1. El visitante presiona el botón *iniciar sesión*, en cualquiera de las dos pantallas principales del sitio público (IUAPI-01 PÁGINA DE INICIO o IUAPI-02 PÁGINA DE DOCUMENTACIÓN).
2. El sistema muestra la pantalla IUAPI-03 FORMULARIO DE INICIO DE SESIÓN con el botón *iniciar sesión* deshabilitado.
3. El visitante introduce su correo electrónico y contraseña; [01A: EL VISITANTE CANCELA LA OPERACIÓN].
4. El sistema habilita el botón *iniciar sesión*.
5. El visitante presiona el botón *iniciar sesión*; [01A: EL VISITANTE CANCELA LA OPERACIÓN].
6. El sistema valida las credenciales del usuario; [01B: CREDENCIALES INCORRECTAS, 01C: CORREO NO VERIFICADO, 01D: CLIENTE NO APROBADO, 01E: CLIENTE RECHAZADO, 01F: CLIENTE EN

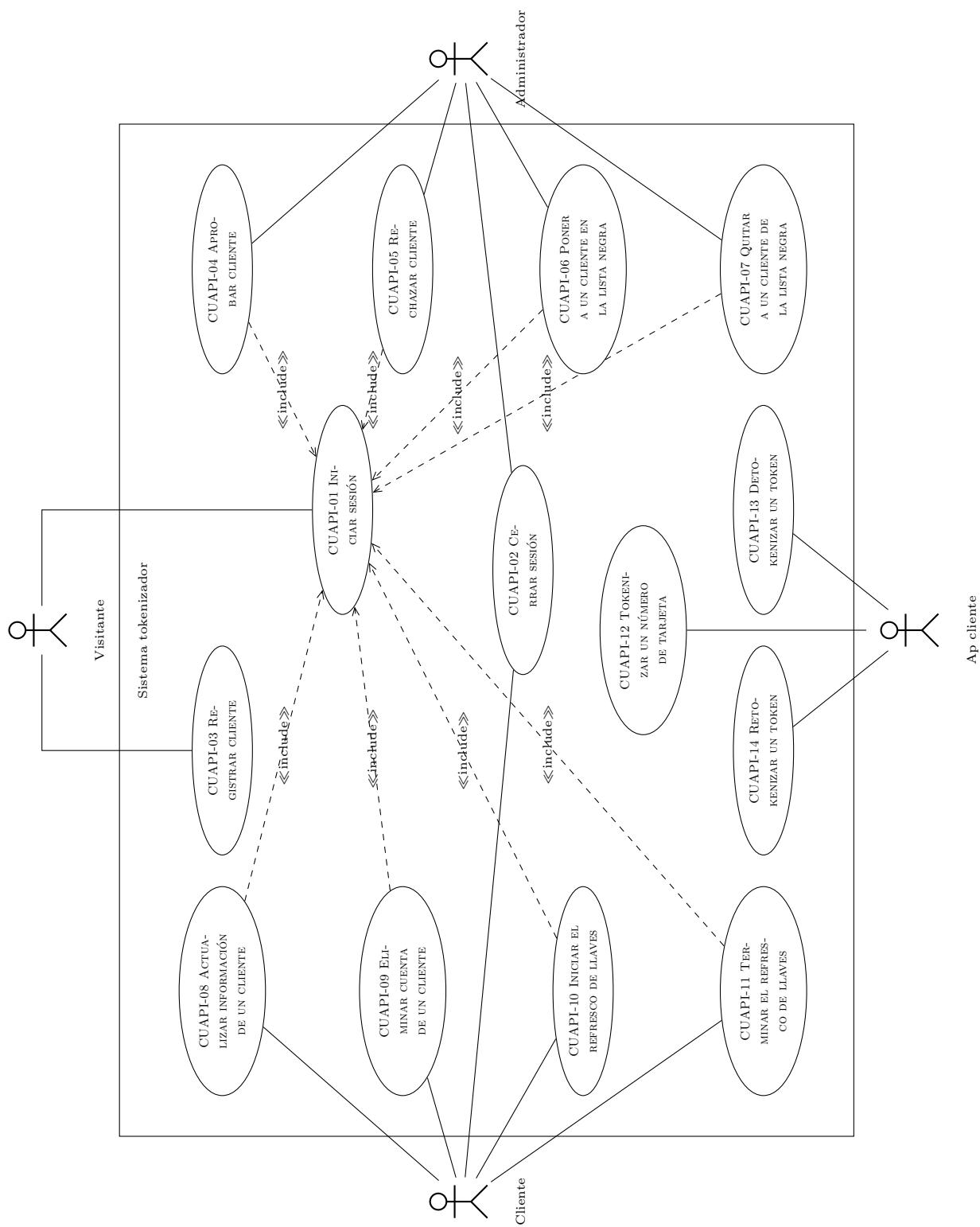


Figura 6.6: Diagrama general de casos de uso.

LISTA NEGRA].

7. El sistema registra al usuario en las variables de sesión.
8. El sistema identifica el tipo de actor que está iniciando sesión; [01G: EL TIPO DE ACTOR ES UN ADMINISTRADOR].
9. El sistema obtiene el estado del cliente.
10. El sistema muestra la pantalla IUAPI-05 PÁGINA DE CONTROL DE CLIENTE tomando en cuenta el estado del cliente.
11. El sistema calcula los días restantes para la expiración de las llaves del cliente (regla de negocios RNAPI-18 CRIPTOPERIODO DE UNA LLAVE); [01H: LAS LLAVES DEL CLIENTE CADUCARON].

Trayectoria alternativa 01A: El visitante cancela la operación.

1. El visitante presiona el botón *cancelar*.
2. El sistema muestra la interfaz de origen del paso 1 de la trayectoria principal.

Trayectoria alternativa 01B: Credenciales incorrectas.

1. El correo electrónico no se encuentra registrado en el sistema o la contraseña dada no corresponde al correo electrónico dado.
2. El sistema muestra el mensaje MSJAPI-01 CREDENCIALES INCORRECTAS.
3. Se regresa al paso 2 de la trayectoria principal.

Trayectoria alternativa 01C: Correo no verificado.

1. El sistema muestra el mensaje MSJAPI-02 CORREO NO VERIFICADO.
2. Se regresa al paso 3 de la trayectoria principal.

Trayectoria alternativa 01D: Cliente no aprobado.

1. El sistema muestra el mensaje MSJAPI-03 CLIENTE NO APROBADO.
2. Se regresa al paso 3 de la trayectoria principal.

Trayectoria alternativa 01E: Cliente rechazado.

1. El sistema muestra el mensaje MSJAPI-04 CLIENTE RECHAZADO.
2. Se regresa al paso 3 de la trayectoria principal.

Trayectoria alternativa 01F: Cliente en lista negra.

1. El sistema muestra el mensaje MSJAPI-05 CLIENTE EN LISTA NEGRA.
2. Se regresa al paso 3 de la trayectoria principal.

Trayectoria alternativa 01G: El tipo de actor es un administrador.

1. El sistema recupera la información de los diez primeros clientes de los siguientes estados:
 - **aprobados o en cambio de llaves.**
 - **verificados.**
 - **en lista negra.**
2. El sistema muestra la pantalla IUAPI-06 PÁGINA DE ADMINISTRACIÓN GENERAL con los datos obtenidos en el paso anterior.

Trayectoria alternativa 01H: Las llaves del cliente caducaron.

1. El sistema muestra el mensaje MSJAPI-06 ADVERTENCIA DE EXPIRACIÓN DE LLAVES.

CUAPI-02 Cerrar sesión.

Permite cerrar la sesión de un **administrador** o de un **cliente**. El actor presiona el botón correspondiente en cualquiera de las interfaces en la que se encuentre y el sistema elimina el registro correspondiente en la sesión.

Trayectoria principal.

1. El **cliente** o **administrador** presiona el botón *cerrar sesión* en cualquiera de las interfaces en la que se encuentre.
2. El sistema elimina el registro correspondiente en las variables de sesión.
3. El sistema muestra la pantalla IUAPI-01 PÁGINA DE INICIO.

CUAPI-03 Registrar cliente.

Caso que permite que un **visitante** se registre y pase a ser un **cliente** del sistema tokenizador (en conformidad con la regla de negocio RNAPI-07 TIPOS DE USUARIO).

Trayectoria principal.

1. El visitante presiona el botón *registrarse*, en cualquiera de las dos pantallas principales del sitio público (IUAPI-01 PÁGINA DE INICIO o IUAPI-02 PÁGINA DE DOCUMENTACIÓN).
2. El sistema muestra la pantalla IUAPI-04 FORMULARIO DE REGISTRO DE USUARIO con el botón *aceptar* deshabilitado.
3. El visitante introduce un correo electrónico y una contraseña (junto con su confirmación); [03A: EL VISITANTE CANCELA LA OPERACIÓN].
4. El sistema habilita el botón *aceptar*.
5. El visitante presiona el botón *aceptar*.

6. El sistema valida el correo electrónico de acuerdo a la regla RNAPI-04 FORMATO DE CORREOS ELECTRÓNICOS; [03B: CORREO ELECTRÓNICO INCORRECTO].
7. El sistema valida el formato de la contraseña de acuerdo a la regla RNAPI-05 FORMATO DE CONTRASEÑAS; [03C: CONTRASEÑA INCORRECTA].
8. El sistema valida que la confirmación de la contraseña coincida con la contraseña introducida; [03D: CONFIRMACIÓN DE CONTRASEÑA INCORRECTA].
9. El sistema valida que el correo introducido no se encuentre ya asignado a un usuario tomando en cuenta la regla de negocios RNAPI-03 UNICIDAD DEL CORREO ELECTRÓNICO; [03E: CORREO REGISTRADO PREVIAMENTE].
10. El sistema crea un nuevo cliente en estado **en espera**, según la regla de negocios RNAPI-08 ESTADOS DE UN CLIENTE, un nuevo correo en estado **no verificado**, según la regla RNAPI-11 ESTADOS DE UN CORREO ELECTRÓNICO, y un nuevo vínculo con la fecha actual, según RNAPI-20 HIPERVÍNCULO DE VERIFICACIÓN.
11. El sistema manda un correo al cliente con el hipervínculo de verificación.
12. El sistema muestra la ventana IUAPI-07 VENTANA DE AVISO DE CORREO DE CONFIRMACIÓN.
13. El cliente accede a la UNIFORM RESOURCE LOCATOR (URL) enviada a su correo.
14. El sistema verifica, mediante la fecha de creación del vínculo, que no haya expirado, tomando en cuenta la regla de negocios RNAPI-21 EXPIRACIÓN DE VÍNCULOS DE VERIFICACIÓN DE REGISTRO; [03F: VÍNCULO EXPIRADO].
15. El sistema cambia el estado del correo a **verificado**, según la regla de negocios RNAPI-11 ESTADOS DE UN CORREO ELECTRÓNICO.
16. El sistema muestra la ventana IUAPI-09 VENTANA DE AVISO DE ESPERA PARA APROBACIÓN.

Trayectoria alternativa 03A: El visitante cancela la operación.

1. El visitante presiona el botón *cancelar*.
2. El sistema muestra la interfaz de origen del paso 1 de la trayectoria principal.

Trayectoria alternativa 03B: Correo electrónico incorrecto.

1. El sistema muestra el mensaje MSJAPI-07 CORREO ELECTRÓNICO INCORRECTO.
2. Se regresa al paso 3 de la trayectoria principal.

Trayectoria alternativa 03C: Contraseña incorrecta.

1. El sistema muestra el mensaje MSJAPI-08 CONTRASEÑA INCORRECTA.
2. Se regresa al paso 3 de la trayectoria principal.

Trayectoria alternativa 03D: Confirmación de contraseña incorrecta.

1. El sistema muestra el mensaje MSJAPI-09 CONFIRMACIÓN DE CONTRASEÑA INCORRECTA.
2. Se regresa al paso 3 de la trayectoria principal.

Trayectoria alternativa 03E: Correo registrado previamente.

1. El sistema muestra el mensaje MSJAPI-10 CORREO REGISTRADO PREVIAMENTE.
2. Se regresa al paso 3 de la trayectoria principal.

Trayectoria alternativa 03F: Vínculo expirado.

1. El sistema muestra la ventana IUAPI-08 VENTANA DE AVISO DE EXPIRACIÓN DE HIPERVÍNCULO.
2. El sistema elimina el registro del usuario.

CUAPI-04 Aprobar cliente.

Caso de uso que permite a un usuario tipo **administrador** aprobar a un usuario tipo **cliente**, haciéndolo pasar del estado **en espera** a **aprobado** (véase RNAPI-08 ESTADOS DE UN CLIENTE).

Trayectoria principal.

1. El administrador presiona el botón *Aprobar* del cliente que desea aprobar en la interfaz IUAPI-06 PÁGINA DE ADMINISTRACIÓN GENERAL.
2. El sistema cambia el estado de dicho cliente a **aprobado** tomando en cuenta la regla de negocios RNAPI-08 ESTADOS DE UN CLIENTE.
3. El sistema manda un correo al usuario notificándole de su aprobación en el sistema.
4. El sistema crea un nuevo juego de llaves con estado **actual** y las asocia al cliente aprobado (regla de negocios RNAPI-10 ESTADOS DE UNA LLAVE).
5. El sistema refresca la interfaz IUAPI-06 PÁGINA DE ADMINISTRACIÓN GENERAL.

CUAPI-05 Rechazar cliente.

Permite a un usuario tipo **administrador** rechazar a un usuario tipo **cliente**, haciéndolo pasar del estado **en espera** a **rechazado** (véase RNAPI-08 ESTADOS DE UN CLIENTE).

Trayectoria principal.

1. El administrador presiona el botón *Rechazar* del cliente que desea rechazar en la interfaz IUAPI-06 PÁGINA DE ADMINISTRACIÓN GENERAL.
2. El sistema muestra el mensaje MSJAPI-13 RECHAZAR CLIENTE.
3. El administrador presiona *Aceptar*; [05A: CANCELAR OPERACIÓN].

4. El sistema cambia el estado de dicho cliente a **rechazado** tomando en cuenta la regla de negocios RNAPI-08 ESTADOS DE UN CLIENTE.
5. El sistema manda un correo al usuario notificándole de su rechazo en el sistema.
6. El sistema refresca la interfaz IUAPI-06 PÁGINA DE ADMINISTRACIÓN GENERAL.

Trayectoria alternativa 05A: Cancelar operación.

1. El administrador presiona *Cancelar*.

CUAPI-06 Poner a un cliente en la lista negra.

Permite a un usuario tipo **administrador** poner a un **cliente** en la lista negra, cambiando su estado a **en lista negra** (véase RNAPI-08 ESTADOS DE UN CLIENTE).

Trayectoria principal.

1. El administrador presiona el botón *Vetar* correspondiente al cliente que desea vetar en la interfaz IUAPI-06 PÁGINA DE ADMINISTRACIÓN GENERAL.
2. El sistema muestra el mensaje MSJAPI-15 VETAR CLIENTE.
3. El administrador presiona *Aceptar*; [06A: CANCELAR OPERACIÓN].
4. El sistema cambia el estado de dicho cliente a **en lista negra** tomando en cuenta la regla de negocios RNAPI-08 ESTADOS DE UN CLIENTE.
5. El sistema manda un correo para avisar al cliente de su ingreso a la lista negra.
6. El sistema refresca la interfaz IUAPI-06 PÁGINA DE ADMINISTRACIÓN GENERAL.

Trayectoria alternativa 06A: Cancelar operación.

1. El administrador presiona *Cancelar*.

CUAPI-07 Quitar a un cliente de la lista negra.

Permite a un usuario tipo **administrador** quitar a un **cliente** de la lista negra, cambiando su estado a **aprobado** (véase RNAPI-08 ESTADOS DE UN CLIENTE).

Trayectoria principal.

1. El administrador presiona el botón *Desvetar* correspondiente al cliente que desea quitar el voto en la interfaz IUAPI-06 PÁGINA DE ADMINISTRACIÓN GENERAL.
2. El sistema cambia el estado de dicho cliente a **aprobado** tomando en cuenta la regla de negocios RNAPI-08 ESTADOS DE UN CLIENTE.
3. El sistema regresa el contador de malas acciones del usuario a cero tomando en cuenta la regla de negocios [RNAPI-12 CONTADOR DE MALAS ACCIONES, RNAPI-13 SANCIONAR CLIENTE].

4. El sistema manda un correo para avisar al cliente de su salida de la lista negra.
5. El sistema refresca la interfaz IUAPI-06 PÁGINA DE ADMINISTRACIÓN GENERAL.

CUAPI-08 Actualizar información de un cliente.

Permite a un usuario tipo **cliente** actualizar sus datos.

Trayectoria principal.

1. El cliente presiona *Actualizar información* en la interfaz IUAPI-05 PÁGINA DE CONTROL DE CLIENTE.
2. El sistema muestra la interfaz con el botón *aceptar* deshabilitado.
3. El cliente introduce un correo electrónico y una contraseña (junto con su confirmación); [08A: EL VISITANTE CANCELA LA OPERACIÓN].
4. El sistema habilita el botón *aceptar*.
5. El cliente presiona el botón *aceptar*.
6. El sistema valida el correo electrónico de acuerdo a la regla RNAPI-04 FORMATO DE CORREOS ELECTRÓNICOS; [08B: CORREO ELECTRÓNICO INCORRECTO].
7. El sistema valida el formato de la contraseña de acuerdo a la regla RNAPI-05 FORMATO DE CONTRASEÑAS; [08C: CONTRASEÑA INCORRECTA].
8. El sistema valida que la confirmación de la contraseña coincida con la contraseña introducida; [08D: CONFIRMACIÓN DE CONTRASEÑA INCORRECTA].
9. El sistema valida que el correo introducido no se encuentre ya asignado a un usuario, tomando en cuenta la regla de negocios RNAPI-03 UNICIDAD DEL CORREO ELECTRÓNICO; [08E: CORREO REGISTRADO PREVIAMENTE].
10. El sistema registra la nueva dirección del cliente en estado **no verificado**, según la regla de negocios RNAPI-11 ESTADOS DE UN CORREO ELECTRÓNICO y un nuevo vínculo tomando en cuenta la regla de negocios RNAPI-20 HIPERVÍNCULO DE VERIFICACIÓN.
11. El sistema elimina el correo con estado con estado **verificado** (regla de negocio RNAPI-11 ESTADOS DE UN CORREO ELECTRÓNICO).
12. El sistema manda un correo al cliente con un hipervínculo de verificación.
13. El sistema muestra la ventana IUAPI-07 VENTANA DE AVISO DE CORREO DE CONFIRMACIÓN.
14. El cliente accede a la URL enviada a su correo.
15. El sistema verifica el correo, tomando en cuenta la regla de negocios RNAPI-22 EXPIRACIÓN DE VÍNCULOS DE VERIFICACIÓN DE ACTUALIZACIÓN.

16. El sistema cambia el estado del correo del cliente a **verificado**, según la regla de negocios RNAPI-11 ESTADOS DE UN CORREO ELECTRÓNICO.
17. El sistema muestra la ventana IUAPI-10 VENTANA DE AVISO DE VERIFICACIÓN EXITOSA.

Trayectoria alternativa 08A: El visitante cancela la operación.

1. El visitante presiona el botón *cancelar*.
2. El sistema muestra la interfaz de origen del paso 1 de la trayectoria principal.

Trayectoria alternativa 08B: Correo electrónico incorrecto.

1. El sistema muestra el mensaje MSJAPI-07 CORREO ELECTRÓNICO INCORRECTO.
2. Se regresa al paso 3 de la trayectoria principal.

Trayectoria alternativa 08C: Contraseña incorrecta.

1. El sistema muestra el mensaje MSJAPI-08 CONTRASEÑA INCORRECTA.
2. Se regresa al paso 3 de la trayectoria principal.

Trayectoria alternativa 08D: Confirmación de contraseña incorrecta.

1. El sistema muestra el mensaje MSJAPI-09 CONFIRMACIÓN DE CONTRASEÑA INCORRECTA.
2. Se regresa al paso 3 de la trayectoria principal.

Trayectoria alternativa 08E: Correo registrado previamente.

1. El sistema muestra el mensaje MSJAPI-10 CORREO REGISTRADO PREVIAMENTE.
2. Se regresa al paso 3 de la trayectoria principal.

CUAPI-09 Eliminar cuenta de un cliente.

Permite a un usuario tipo **cliente** eliminar su cuenta.

Trayectoria principal.

1. El cliente presiona *eliminar cuenta* en la interfaz IUAPI-05 PÁGINA DE CONTROL DE CLIENTE.
2. El sistema muestra el mensaje MSJAPI-21 ADVERTENCIA AL ELIMINAR UN CLIENTE.
3. El cliente presiona el botón *aceptar*; [09A: EL CLIENTE CANCELA LA OPERACIÓN].
4. El sistema elimina el registro del cliente en la base de datos (RNAPI-11 ESTADOS DE UN CORREO ELECTRÓNICO).
5. El sistema elimina todos los tokens (RNAPI-09 ESTADOS DE UN TOKEN) y las llaves (RNAPI-10 ESTADOS DE UNA LLAVE) relacionadas con el cliente.

6. El sistema elimina las variables del sesión del cliente.
7. El sistema muestra la interfaz IUAPI-01 PÁGINA DE INICIO.

Trayectoria alternativa 09A: El cliente cancela la operación.

1. El cliente presiona el botón *cancelar*.
2. El sistema muestra la interfaz IUAPI-05 PÁGINA DE CONTROL DE CLIENTE.

CUAPI-10 Iniciar el refresco de llaves.

Permite a un usuario tipo **cliente** iniciar el refresco de llaves.

Trayectoria principal.

1. El cliente presiona *Iniciar refresco de llaves* en la interfaz IUAPI-05 PÁGINA DE CONTROL DE CLIENTE.
2. El sistema cambia el estado de todos los tokens relacionados a él, con estado **actual** a **anterior** (regla de negocio RNAPI-09 ESTADOS DE UN TOKEN).
3. El sistema cambia el estado de todas las llaves asociadas con el usuario en estado **actual** a **anterior** (regla de negocios RNAPI-10 ESTADOS DE UNA LLAVE).
4. El sistema crea un juego nuevo de llaves con estado **actual** y las asocia al cliente (regla de negocios RNAPI-10 ESTADOS DE UNA LLAVE).
5. El sistema cambia el estado del usuario a **en cambio de llaves** (regla de negocios RNAPI-08 ESTADOS DE UN CLIENTE).

CUAPI-11 Terminar el refresco de llaves.

Permite a un usuario tipo **cliente** terminar el refresco de llaves.

Trayectoria principal.

1. El cliente presiona *Terminar refresco de llaves* en la interfaz IUAPI-05 PÁGINA DE CONTROL DE CLIENTE.
2. El sistema verifica que no haya tokens relacionados con el cliente que se encuentren en estado **anterior**; [11A: EL CLIENTE AÚN TIENE TOKENS RELACIONADOS CON ESTADO ANTERIOR].
3. El sistema elimina todas las llaves asociadas al cliente con estado **anterior** (regla de negocios RNAPI-10 ESTADOS DE UNA LLAVE).
4. El sistema elimina todos los tokens cuyo estado sea **retokenizado** (regla de negocio RNAPI-09 ESTADOS DE UN TOKEN).
5. El sistema cambia el estado del cliente a **aprobado** (regla de negocios RNAPI-08 ESTADOS DE UN CLIENTE).

Trayectoria alternativa 11A: El cliente aún tiene tokens relacionados con estado anterior.

1. El sistema muestra el mensaje MSJAPI-20 ADVERTENCIA DE RETOKENIZACIÓN INCOMPLETA.
2. El usuario presiona *Aceptar*; [11B: EL CLIENTE CANCELA EL TERMINAR CON EL REFRESCO DE LLAVES].
3. El sistema elimina todos los tokens relacionados con el cliente que se encuentran en estado **anterior** (regla de negocio RNAPI-09 ESTADOS DE UN TOKEN).
4. El sistema regresa al paso 3 de la trayectoria principal.

Trayectoria alternativa 11B: El cliente cancela el terminar con el refresco de llaves.

1. El usuario presiona *Cancelar*.
2. El sistema muestra la interfaz de inicio .

CUAPI-12 Tokenizar un número de tarjeta.

Permite a un usuario tipo **aplicación cliente**, dada una conexión establecida, obtener el token de una tarjeta.

Trayectoria principal.

1. La aplicación cliente envía una petición con los siguientes datos:
 - Número de tarjeta a tokenizar.
 - Algoritmo tokenizador.
2. El sistema obtiene el estado del cliente.
3. El sistema verifica que el estado del cliente sea **aceptado** o **en cambio de llaves** (tomando en cuenta la regla de negocios RNAPI-07 TIPOS DE USUARIO); [12A: EL CLIENTE TIENE UN ESTADO DISTINTO A **ACEPTADO O EN CAMBIO DE LLAVES**].
4. El sistema verifica que el tipo de algoritmo utilizado sea **irreversible** (tomando en cuenta la regla de negocios RNAPI-06 TIPOS DE ALGORITMOS TOKENIZADORES); [12B: EL ALGORITMO ESPECIFICADO POR EL CLIENTE ES DE TIPO **REVERSIBLE**].
5. El sistema verifica que la tarjeta no tenga asociado un token con ese cliente; [12C: EL CLIENTE YA TIENE UN TOKEN RELACIONADO CON ESA TARJETA].
6. El sistema obtiene el token utilizando el algoritmo especificado y la llave asociada al algoritmo con estado **actual**, tomando en cuenta la regla de negocios RNAPI-14 PETICIÓN DE TOKENIZACIÓN.
7. El sistema registra el token obtenido en el paso anterior en la base de datos con estado **actual** tomando en cuenta la regla de negocios RNAPI-09 ESTADOS DE UN TOKEN.
8. El sistema regresa a la aplicación cliente el token generado.

Trayectoria alternativa 12A: El cliente tiene un estado distinto a aceptado o en cambio de llaves.

1. El sistema regresa una respuesta HYPERTEXT TRANSFER PROTOCOL (HTTP) con código 403.

Trayectoria alternativa 12B: El algoritmo especificado por el cliente es de tipo reversible.

1. El sistema obtiene el token con el algoritmo especificado utilizando la llave con estado **actual**, tomando en cuenta la regla de negocios RNAPI-14 PETICIÓN DE TOKENIZACIÓN.
2. Regresar al paso 8 de la trayectoria principal.

Trayectoria alternativa 12C: El cliente ya tiene un token relacionado con esa tarjeta.

1. El sistema regresa una respuesta HTTP con código 403.

CUAPI-13 Detokenizar un TOKEN.

Permite a un usuario tipo **aplicación cliente**, dada una conexión establecida, obtener el número de una tarjeta a partir de un token.

Trayectoria principal.

1. La aplicación cliente envía una petición con los siguientes datos:
 - Token.
 - Algoritmo tokenizador.
2. El sistema obtiene el estado del cliente.
3. El sistema verifica que el estado del cliente sea **aceptado o en cambio de llaves**, (tomando en cuenta la regla de negocios RNAPI-08 ESTADOS DE UN CLIENTE); [13A: EL CLIENTE TIENE UN ESTADO DISTINTO A **ACEPTADO O EN CAMBIO DE LLAVES**].
4. El sistema verifica que el token dado sea válido (regla de negocios RNAPI-02 COMPOSICIÓN DE TOKEN); [13E: TOKEN CON FORMATO INVÁLIDO].
5. El sistema verifica que el tipo de algoritmo utilizado sea **irreversible** (tomando en cuenta la regla de negocios RNAPI-06 TIPOS DE ALGORITMOS TOKENIZADORES); [13B: EL CLIENTE SE ENCUENTRA EN ESTADO **ACEPTADO** Y EL ALGORITMO ESPECIFICADO ES DE TIPO **REVERSIBLE**, 13C: EL CLIENTE SE ENCUENTRA EN ESTADO **EN CAMBIO DE LLAVES** Y EL ALGORITMO ESPECIFICADO ES DE TIPO **REVERSIBLE**].
6. El sistema obtiene el número de tarjeta relacionado con este token realizando una consulta a la base de datos; [13D: EL TOKEN ESPECIFICADO NO SE ENCUENTRA REGISTRADO EN LA BASE DE DATOS].
7. El sistema regresa a la aplicación cliente el número de tarjeta obtenido en el paso anterior.

Trayectoria alternativa 13A: El cliente tiene un estado distinto a aceptado o en cambio de llaves.

1. El sistema regresa una respuesta HTTP con código 403.

Trayectoria alternativa 13B: El cliente se encuentra en estado aceptado y el algoritmo especificado es de tipo reversible.

1. El sistema obtiene el número de tarjeta con el token y la llave con estado **actual**, tomando en cuenta la regla de negocios RNAPI-15 PETICIÓN DE DETOKENIZACIÓN.
2. Regresar al paso 7 de la trayectoria principal.

Trayectoria alternativa 13C: El cliente se encuentra en estado en cambio de llaves y el algoritmo especificado es de tipo reversible.

1. El sistema obtiene de la petición la llave con la que se realizará la operación de detokenización, tomando en cuenta la regla de negocios RNAPI-16 PETICIÓN DE DETOKENIZACIÓN EN PROCESO DE REFRESCO.
2. El sistema obtiene el número de tarjeta con el token y la llave especificada en el paso anterior.
3. Regresar al paso 7 de la trayectoria principal.

Trayectoria alternativa 13D: EL token especificado no se encuentra registrado en la base de datos.

1. El sistema incrementa el contador de malas acciones del usuario en tres unidades (regla de negocios RNAPI-12 CONTADOR DE MALAS ACCIONES).
2. El sistema verifica que el contador de malas acciones del usuario no sea igual o mayor al umbral permitido (RNAPI-13 SANCIONAR CLIENTE); [13F: LÍMITE DE MALAS ACCIONES ALCANZADO].
3. El sistema regresa una respuesta HTTP con código 400.

Trayectoria alternativa 13E: Token con formato inválido.

1. El sistema incrementa el contador de malas acciones del usuario en una unidad (regla de negocios RNAPI-12 CONTADOR DE MALAS ACCIONES).
2. El sistema verifica que el contador de malas acciones del usuario no sea igual o mayor al umbral permitido (RNAPI-13 SANCIONAR CLIENTE); [13F: LÍMITE DE MALAS ACCIONES ALCANZADO].
3. El sistema regresa una respuesta HTTP con código 400.

Trayectoria alternativa 13F: Límite de malas acciones alcanzado.

1. El sistema cambia el estado del cliente a **en lista negra**.
2. El sistema regresa una respuesta HTTP con código 400.

CUAPI-14 Retokenizar un TOKEN.

Permite a un usuario tipo **aplicación cliente**, dada una conexión establecida, obtener el token nuevo a partir de otro token.

Trayectoria principal.

1. La aplicación cliente envía una petición con los siguientes datos:
 - Token a retokenizar.
 - Algoritmo tokenizador.
2. El sistema obtiene el estado del cliente.
3. El sistema verifica que el estado del cliente sea **en cambio de llaves** (tomando en cuenta la regla de negocios RNAPI-08 ESTADOS DE UN CLIENTE); [14A: EL CLIENTE TIENE UN ESTADO DISTINTO A **EN CAMBIO DE LLAVES**].
4. El sistema verifica que el token dado sea válido (regla de negocios RNAPI-02 COMPOSICIÓN DE TOKEN); [14D: TOKEN CON FORMATO INVÁLIDO].
5. El sistema verifica que el tipo de algoritmo utilizado sea **irreversible** (tomando en cuenta la regla de negocios RNAPI-06 TIPOS DE ALGORITMOS TOKENIZADORES); [14B: EL ALGORITMO ESPECIFICADO POR EL CLIENTE ES DE TIPO **REVERSIBLE**].
6. El sistema verifica que el estado del token sea **anterior** o **retokenizado**, tomando en cuenta la regla de negocios RNAPI-09 ESTADOS DE UN TOKEN; [14C: EL TOKEN ESPECIFICADO NO SE ENCUENTRA REGISTRADO EN LA BASE DE DATOS].
7. El sistema obtiene el número de tarjeta relacionado con este token realizando una consulta a la base de datos.
8. El sistema obtiene el token correspondiente al número de tarjeta obtenido en el paso anterior utilizando el algoritmo especificado y la llave asociada al algoritmo con estado **actual**, tomando en cuenta la regla de negocios RNAPI-14 PETICIÓN DE TOKENIZACIÓN.
9. El sistema registra el token obtenido en el paso anterior en la base de datos con estado **actual** tomando en cuenta la regla de negocios RNAPI-09 ESTADOS DE UN TOKEN.
10. El sistema cambia el estado del token recibido en la petición a **retokenizado**.
11. El sistema regresa a la aplicación cliente el token generado.

Trayectoria alternativa 14A: El cliente tiene un estado distinto a en cambio de llaves.

1. El sistema regresa una respuesta HTTP con código 403.

Trayectoria alternativa 14B: El algoritmo especificado por el cliente es de tipo reversible.

1. El sistema obtiene el número de tarjeta con el token y la llave asociada al algoritmo con estado **anterior**.
2. El sistema obtiene el token correspondiente al número de tarjeta mediante el algoritmo especificado y la llave asociada con estado **actual**.
3. Regresar al paso 11 de la trayectoria principal.

Trayectoria alternativa 14C: EL token especificado no se encuentra registrado en la base de datos.

1. El sistema incrementa el contador de malas acciones del usuario en tres unidades (regla de negocios RNAPI-12 CONTADOR DE MALAS ACCIONES).
2. El sistema verifica que el contador de malas acciones del usuario no sea igual o mayor al umbral permitido (RNAPI-13 SANCIONAR CLIENTE); [14E: LÍMITE DE MALAS ACCIONES ALCANZADO].
3. El sistema regresa una respuesta HTTP con código 400.

Trayectoria alternativa 14D: Token con formato inválido.

1. El sistema incrementa el contador de malas acciones del usuario en una unidad (regla de negocios RNAPI-12 CONTADOR DE MALAS ACCIONES).
2. El sistema verifica que el contador de malas acciones del usuario no sea igual o mayor al umbral permitido (RNAPI-13 SANCIONAR CLIENTE); [14E: LÍMITE DE MALAS ACCIONES ALCANZADO].
3. El sistema regresa una respuesta HTTP con código 400.

Trayectoria alternativa 14E: Límite de malas acciones alcanzado.

1. El sistema cambia el estado del cliente a **en lista negra**.
2. El sistema regresa una respuesta HTTP con código 400.

6.1.4. Interfaces de usuario

6.1.5. Catálogo de mensajes

MSJAPI-01 Credenciales incorrectas.

Nombre de usuario o contraseña incorrectos.

MSJAPI-02 Correo no verificado.

Este correo no está verificado; verifíquelo haciendo clic en el enlace que le fue mandando (revise su carpeta de SPAM).

MSJAPI-03 Cliente no aprobado.

Capítulo 6. Análisis y diseño de aplicación web

The screenshot shows the homepage of the Sistema Tokenizador. At the top, there's a navigation bar with 'Sistema Tokenizador' on the left, 'REGISTRARSE' and 'INICIAR SESIÓN' on the right, and a user icon in the center. Below the navigation is a sidebar with 'INICIO' and 'DOCUMENTACIÓN' buttons. The main content area has a title '¿Qué es la tokenización?' followed by two sections: 'Primero, ¿qué es un token?' and 'Sobre la tokenización'. Both sections contain detailed text about tokens and their properties.

(a) 1920 x 1080

This screenshot shows the same homepage layout as (a), but with a smaller window size of 1280x800. The content is slightly compressed, and the sidebar icons are smaller. The main text sections are also scaled down.

(b) 1280 x 800

(c) 480 x 800

This screenshot shows the homepage at a very low resolution of 480x800. The text is extremely small and difficult to read, and the overall layout appears compressed and less readable.

(d) 600 x 960

(e) 960 x 900

IUAPI-01 Página de inicio

Generación de tokens para proteger los datos de tarjetas bancarias

This screenshot shows a web browser displaying the 'Documentación' (Documentation) section of a tokenization API. The page has a dark header with 'Sistema Tokenizador' and 'Documentación'. It includes 'REGISTRARSE' and 'INICIAR SESIÓN' buttons. The main content area contains sections for 'Cómo utilizar la API web', 'Tokenización', 'Uso y ejemplo', and 'Errores'. The 'Uso y ejemplo' section shows a JSON example for tokenizing a card with PAN '28045809693113314' and method 'FFX'. The 'Errores' section lists various error codes and their descriptions.

(a) 1920 x 1080

This screenshot shows the same documentation page as above, but at a lower resolution of 1280x800. The layout is similar, with the 'Documentación' header, 'REGISTRARSE', and 'INICIAR SESIÓN' buttons. The content sections ('Cómo utilizar la API web', 'Tokenización', 'Uso y ejemplo', 'Errores') are present, though some text is less readable due to the smaller font size.

(b) 1280 x 800

(c) 480 x 800

This screenshot shows the documentation page at a resolution of 600x960. The content is significantly compressed, with the 'Uso y ejemplo' and 'Errores' sections being completely illegible. The overall layout remains consistent with the higher resolution versions.

(d) 600 x 960

(e) 960 x 900

The screenshot shows the homepage of a tokenization service. At the top right are 'REGISTRARSE' and 'INICIAR SESIÓN' buttons. Below them is a modal window titled 'Iniciar sesión' with fields for 'Correo electrónico' and 'Contraseña'. At the bottom of the modal are 'CANCELAR' and 'INICIAR SESIÓN' buttons.

¿Qué es la tokenización?

Un token es un valor parecido y representativo de otro valor, pero que no tiene una relación directa con el mismo; razón por la cual es utilizado para proteger información confidencial. Con la definición de un token, se puede ver a la tokenización como el proceso de sustituir datos sensibles por un token, con la finalidad de mantener su confidencialidad.

¿Por qué usar nuestro servicio de tokenización?

Nuestro servicio de tokenización te permite despacharte de la protección de los datos bancarios sensibles de tu sistema, delegándonos esta labor.

Entre las ventajas que obtienes al usar nuestro servicio de tokenización:

- ✓ Mantener la seguridad de tu sistema cumpliendo con el estándar de seguridad del PCI SSC.
- ✓ Salvaguardar la confidencialidad de datos sensibles por medio de la tokenización.
- ✓ Elegir el algoritmo con el que deseas tokenizar tu información de entre FFX, BPS, TKR, AHR o mediante DRGB.

¿Qué es el PCI SSC?

El Payment Card Industry Security Standards Council (PCI SSC) es una organización internacional encargada de estandarizar, desarrollar e informar sobre cómo hacer transacciones bancarias seguras. Esta organización desarrolló el PCI Data Security Standard (DSS) como un estándar en el que se indica la manera de mantener la información bancaria segura, mediante la implementación de protocolos de seguridad.

Es importante resaltar que el PCI SSC ha establecido como requerimiento la implementación del PCI DSS para todos los comercios que realizan un número de transacciones considerables.

Sobre los algoritmos de tokenización

Nuestro sistema permite seleccionar el algoritmo con el que se desea tokenizar un dato. Contamos con 2 algoritmos de tokenización reversibles, como son FFX y BPS, y 3 algoritmos de tokenización irreversibles, que son TKR, AHR y DRBG. A continuación puedes leer una breve descripción de estos.

(a) 1920 x 1080

This screenshot is identical to the one above, showing the homepage and login modal. The only difference is the resolution, which is 1280x800.

(b) 1280 x 800

(c) 480 x 800

This screenshot is identical to the ones above, showing the homepage and login modal. The only difference is the resolution, which is 600x960.

(d) 600 x 960

(e) 960 x 900

IUAPI-03 Formulario de inicio de sesión

Generación de tokens para proteger los datos de tarjetas bancarias

The screenshot shows the 'Sistema Tokenizador' (Tokenization System) interface. At the top right are 'REGISTRARSE' and 'INICIAR SESIÓN' buttons. The main content area has a dark header 'Inicio'. Below it is a section titled '¿Qué es la tokenización?' with a detailed explanation. A modal window titled 'Registrar datos' is open, prompting for 'Correo electrónico', 'Contraseña', and 'Confirmación de contraseña'. At the bottom of the modal are 'CANCELAR' and 'ACEPTAR' buttons.

(a) 1920 x 1080

This screenshot shows the same system interface as above, but at a lower resolution (1280x800). The layout is slightly compressed, and the text is smaller. The 'Registrar datos' modal is still present, and the overall design is consistent with the higher resolution version.

(b) 1280 x 800

(c) 480 x 800

This screenshot shows the system interface at a resolution of 600x960. The text is significantly smaller and less readable compared to the higher resolutions. The 'Registrar datos' modal is still visible, though its contents are harder to discern due to the low resolution.

(d) 600 x 960

(e) 960 x 900

IUAPI-04 Formulario de registro de usuario

Capítulo 6. Análisis y diseño de aplicación web

The screenshot shows the main landing page of the application. At the top right are 'REGISTRARSE' and 'INICIAR SESIÓN' buttons. On the left, there are 'INICIO' and 'DOCUMENTACIÓN' links. The main content area has several sections: '¿Qué es la tokenización?' with a detailed description; '¿Por qué usar nuestro servicio de tokenización?' with a bulleted list of benefits; '¿Qué es el PCI SSC?' with a brief description; and 'Sobre los algoritmos de tokenización' with another bulleted list. A central modal window titled 'Iniciar sesión' is open, prompting for 'Correo electrónico' and 'Contraseña'.

(a) 1920 x 1080

This screenshot shows the same homepage layout but at a smaller resolution (1280x800). The overall structure is identical to the 1920x1080 version, but the text and some UI elements appear slightly smaller or less sharp.

(b) 1280 x 800

This screenshot shows the homepage at a very low resolution (480x800). The text is significantly smaller and less readable. The 'Iniciar sesión' modal is still present but looks like a simple rectangle.

(c) 480 x 800

This screenshot shows the homepage at a resolution of 600x960. The text is larger than in the 480x800 version but smaller than in the 1920x1080 version. The 'Iniciar sesión' modal is clearly visible.

(d) 600 x 960

This screenshot shows the homepage at a resolution of 960x900. The text size is intermediate between the 600x960 and 1920x1080 versions. The 'Iniciar sesión' modal is clearly visible.

(e) 960 x 900

IUAPI-05 Página de control de cliente

Generación de *tokens* para proteger los datos de tarjetas bancarias

Sistema Tokenizador

Administración

administrador@prueba.com CERRAR SESIÓN

Clientes en espera

Correo electrónico	APROBAR	RECHAZAR
cliente-en-espera-a@prueba.com		
cliente-en-espera-b@prueba.com		
cliente-en-espera-c@prueba.com		
cliente-en-espera-d@prueba.com		
cliente-en-espera-e@prueba.com		

Página: 1 Filas por página: 5 1 - 5 de 20 < >

Clientes aprobados

Correo electrónico	VETAR
cliente-aprobado-b@prueba.com	
cliente-aprobado-c@prueba.com	
cliente-aprobado-d@prueba.com	
cliente-aprobado-e@prueba.com	
cliente-aprobado-f@prueba.com	

Página: 1 Filas por página: 5 1 - 5 de 19 < >

Clientes en lista negra

Correo electrónico

(a) 1920 x 1080

Sistema Tokenizador

Administración

administrador@prueba.com CERRAR SESIÓN

Clientes en espera

Correo electrónico	APROBAR	RECHAZAR
cliente-en-espera-a@prueba.com		
cliente-en-espera-b@prueba.com		
cliente-en-espera-c@prueba.com		
cliente-en-espera-d@prueba.com		
cliente-en-espera-e@prueba.com		

Página: 1 Filas por página: 5 1 - 5 de 20 < >

Clientes aprobados

Correo electrónico	VETAR
cliente-aprobado-b@prueba.com	
cliente-aprobado-c@prueba.com	
cliente-aprobado-d@prueba.com	

Página: 1 Filas por página: 5 1 - 5 de 19 < >

Clientes en lista negra

Correo electrónico

(b) 1280 x 800

(c) 480 x 800

Administración

administrador@prueba.com CERRAR SESIÓN

Clientes en espera

Correo electrónico	APROBAR	RECHAZAR
cliente-en-espera-a@prueba.com		
cliente-en-espera-b@prueba.com		
cliente-en-espera-c@prueba.com		
cliente-en-espera-d@prueba.com		
cliente-en-espera-e@prueba.com		

Página: 1 Filas por página: 5 1 - 5 de 20 < >

Clientes aprobados

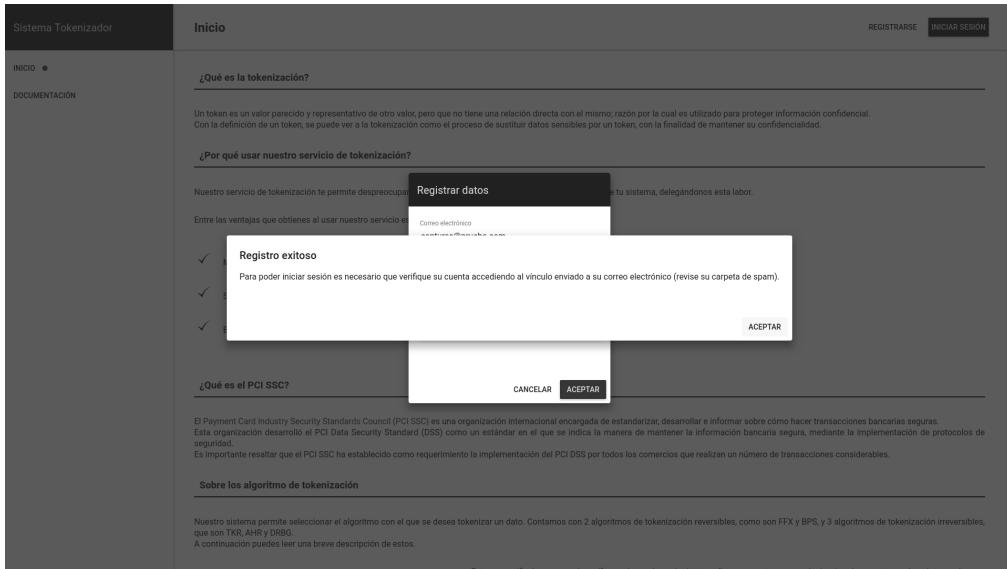
Correo electrónico	VETAR
cliente-aprobado-b@prueba.com	
cliente-aprobado-c@prueba.com	
cliente-aprobado-d@prueba.com	
cliente-aprobado-e@prueba.com	
cliente-aprobado-f@prueba.com	

Página: 1 Filas por página: 5 1 - 5 de 20 < >

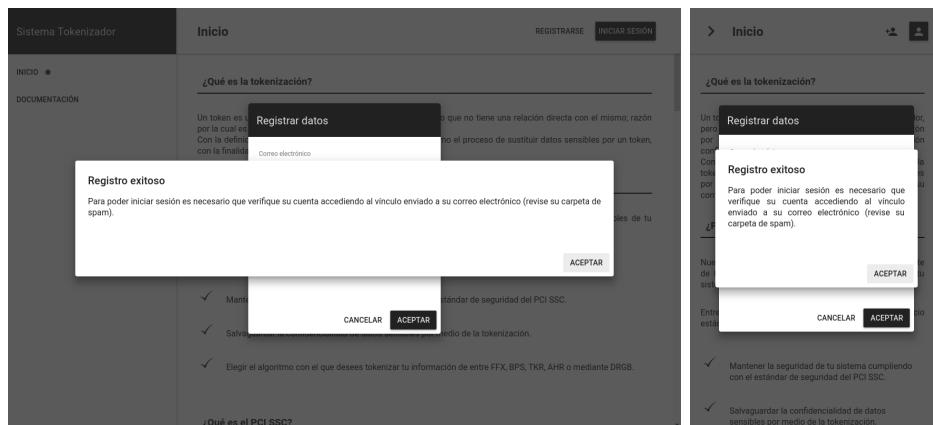
(d) 600 x 960

(e) 960 x 900

IUAPI-06 Página de administración general

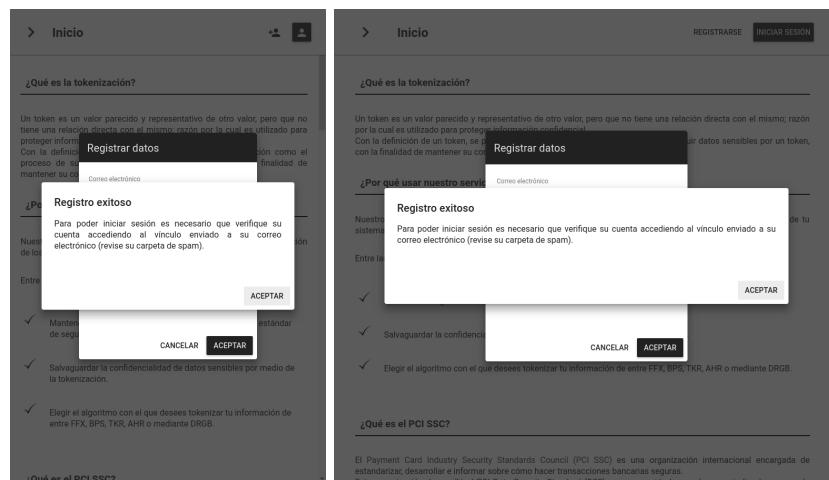


(a) 1920 x 1080



(b) 1280 x 800

(c) 480 x 800



(d) 600 x 960 (e) 960 x 900

IUAPI-07 Ventana de aviso de correo de confirmación

Generación de *tokens* para proteger los datos de tarjetas bancarias

The screenshot shows the main dashboard of the Sistema Tokenizador application. It features a sidebar with 'Sistema Tokenizador' at the top, followed by 'INICIO' and 'DOCUMENTACIÓN'. The main content area has a header 'Inicio' with 'REGISTRARSE' and 'INICIAR SESIÓN' buttons. Below this, there are three main sections: '¿Qué es la tokenización?' (What is tokenization?), '¿Por qué usar nuestro servicio de tokenización?' (Why use our tokenization service?), and '¿Qué es el PCI SSC?'. A modal window titled 'Vínculo expirado' (Link expired) is open, stating 'Han transcurrido más de 24 horas desde su registro. Para verificar su correo vuelva a registrarse.' (More than 24 hours have passed since registration. To verify your email, register again.) with an 'ACEPTAR' (Accept) button.

(a) 1920 x 1080

This screenshot is identical to the one above, showing the main dashboard of the Sistema Tokenizador application. It includes the sidebar, main sections, and the 'Vínculo expirado' (Link expired) modal window.

(b) 1280 x 800

(c) 480 x 800

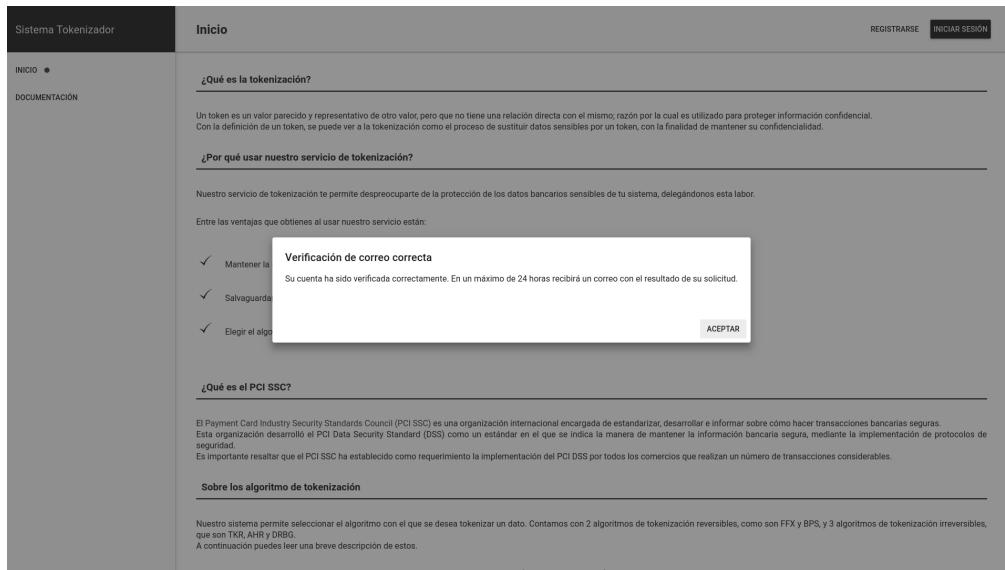
This screenshot is identical to the ones above, showing the main dashboard of the Sistema Tokenizador application. It includes the sidebar, main sections, and the 'Vínculo expirado' (Link expired) modal window.

(d) 600 x 960

(e) 960 x 900

IUAPI-08 Ventana de aviso de expiración de hipervínculo

Capítulo 6. Análisis y diseño de aplicación web



(a) 1920 x 1080

This block contains two screenshots of the Sistema Tokenizador application. Both show the same basic layout as the first screenshot, including the header, sidebar, and main content sections. The difference is in the overall size and readability of the text. The left screenshot is labeled '(b) 1280 x 800' and the right one is labeled '(c) 480 x 800'. Both screenshots include a modal for successful email verification.

(c) 480 x 800

This block contains two screenshots of the Sistema Tokenizador application. The left screenshot is labeled '(d) 600 x 960' and the right one is labeled '(e) 960 x 900'. Both are smaller versions of the previous ones, showing the same content and modal for successful email verification.

(e) 960 x 900

IUAPI-09 Ventana de aviso de espera para aprobación

Generación de *tokens* para proteger los datos de tarjetas bancarias

The screenshot shows the 'Inicio' (Home) page of the Sistema Tokenizador. At the top right are 'REGISTRARSE' and 'INICIAR SESIÓN' buttons. On the left, there are 'INICIO' and 'DOCUMENTACIÓN' links. The main content area has sections like '¿Qué es la tokenización?' and '¿Por qué usar nuestro servicio de tokenización?'. A modal window titled 'Verificación de correo correcta' displays the message 'Su cuenta ha sido verificada correctamente.' with 'ACEPTAR' and 'DRGB' buttons.

(a) 1920 x 1080

This screenshot is identical to (a), showing the same 'Inicio' page and verification message. The only difference is the resolution, which is 1280x800.

(b) 1280 x 800

This screenshot is identical to (a), showing the same 'Inicio' page and verification message. The only difference is the resolution, which is 480x800.

(c) 480 x 800

This screenshot is identical to (a), showing the same 'Inicio' page and verification message. The only difference is the resolution, which is 600x960.

(d) 600 x 960

This screenshot is identical to (a), showing the same 'Inicio' page and verification message. The only difference is the resolution, which is 960x900.

(e) 960 x 900

IUAPI-10 Ventana de aviso de verificación exitosa

Capítulo 6. Análisis y diseño de aplicación web

Sistema Tokenizador

Administración

administrador@prueba.com CERRAR SESIÓN

INICIO
DOCUMENTACIÓN
ADMINISTRACIÓN ●

Clients en espera

Correo electrónico

cliente-en-espera-a@prueba.com	APROBAR	RECHAZAR
cliente-en-espera-b@prueba.com	APROBAR	RECHAZAR
cliente-en-espera-c@prueba.com	APROBAR	RECHAZAR
cliente-en-espera-d@prueba.com	APROBAR	RECHAZAR
cliente-en-espera-e@prueba.com	APROBAR	RECHAZAR

Página: 1 Filas por página: 5 1 - 5 de 20 < >

Clients aprobados

Correo electrónico

cliente-aprobado-b@prueba.com	VETAR
cliente-aprobado-c@prueba.com	VETAR
cliente-aprobado-d@prueba.com	VETAR
cliente-aprobado-e@prueba.com	VETAR
cliente-aprobado-f@prueba.com	VETAR

Página: 1 Filas por página: 5 1 - 5 de 19 < >

Clients en lista negra

Correo electrónico

cliente-en-espera-a@prueba.com	APROBAR	RECHAZAR
cliente-en-espera-b@prueba.com	APROBAR	RECHAZAR
cliente-en-espera-c@prueba.com	APROBAR	RECHAZAR
cliente-en-espera-d@prueba.com	APROBAR	RECHAZAR
cliente-en-espera-e@prueba.com	APROBAR	RECHAZAR

(a) 1920 x 1080

Sistema Tokenizador

Administración

administrador@prueba.com CERRAR SESIÓN

INICIO
DOCUMENTACIÓN
ADMINISTRACIÓN ●

Clients en espera

Correo electrónico

cliente-en-espera-a@prueba.com	APROBAR	RECHAZAR
cliente-en-espera-b@prueba.com	APROBAR	RECHAZAR
cliente-en-espera-c@prueba.com	APROBAR	RECHAZAR
cliente-en-espera-d@prueba.com	APROBAR	RECHAZAR
cliente-en-espera-e@prueba.com	APROBAR	RECHAZAR

Página: 1 Filas por página: 5 1 - 5 de 20 < >

Clients aprobados

Correo electrónico

cliente-aprobado-b@prueba.com	VETAR
cliente-aprobado-c@prueba.com	VETAR
cliente-aprobado-d@prueba.com	VETAR

Página: 1 Filas por página: 5 1 - 5 de 19 < >

Clients en lista negra

Correo electrónico

cliente-en-espera-a@prueba.com	APROBAR	RECHAZAR
cliente-en-espera-b@prueba.com	APROBAR	RECHAZAR
cliente-en-espera-c@prueba.com	APROBAR	RECHAZAR
cliente-en-espera-d@prueba.com	APROBAR	RECHAZAR
cliente-en-espera-e@prueba.com	APROBAR	RECHAZAR

(b) 1280 x 800

(c) 480 x 800

Administración

administrador@prueba.com CERRAR SESIÓN

Clients en espera

Correo electrónico

cliente-en-espera-a@prueba.com	APROBAR	RECHAZAR
cliente-en-espera-b@prueba.com	APROBAR	RECHAZAR
cliente-en-espera-c@prueba.com	APROBAR	RECHAZAR
cliente-en-espera-d@prueba.com	APROBAR	RECHAZAR
cliente-en-espera-e@prueba.com	APROBAR	RECHAZAR

Página: 1 Filas por página: 5 1 - 5 de 20 < >

Clients aprobados

Correo electrónico

cliente-aprobado-b@prueba.com	VETAR
cliente-aprobado-c@prueba.com	VETAR
cliente-aprobado-d@prueba.com	VETAR
cliente-aprobado-e@prueba.com	VETAR
cliente-aprobado-f@prueba.com	VETAR

Página: 1 Filas por página: 5 1 - 5 de 20 < >

Clients en lista negra

Correo electrónico

cliente-en-espera-a@prueba.com	APROBAR	RECHAZAR
cliente-en-espera-b@prueba.com	APROBAR	RECHAZAR
cliente-en-espera-c@prueba.com	APROBAR	RECHAZAR
cliente-en-espera-d@prueba.com	APROBAR	RECHAZAR
cliente-en-espera-e@prueba.com	APROBAR	RECHAZAR

(d) 600 x 960

(e) 960 x 900

IUAPI-11 Formulario de edición de datos de un cliente

Su cuenta aún no está aprobada; favor de esperar.

MSJAPI-04 Cliente rechazado.

Su cuenta ha sido rechazada.

MSJAPI-05 Cliente en lista negra.

Su cuenta se encuentra en la lista negra por usos incorrectos.

MSJAPI-06 Advertencia de expiración de llaves.

Atención: sus llaves han expirado. Debe generar nuevas llaves iniciando el proceso de refresco de llaves y retokenizando sus tokens.

MSJAPI-07 Correo electrónico incorrecto.

Correo electrónico inválido (RFC 5322).

MSJAPI-08 Contraseña incorrecta.

Contraseña inválida: deben ser entre 8 y 24 caracteres ASCII imprimibles.

MSJAPI-09 Confirmación de contraseña incorrecta.

La confirmación de contraseña no coincide con la contraseña dada.

MSJAPI-10 Correo registrado previamente.

El correo dado ya se encuentra registrado.

MSJAPI-11 Sin clientes para mostrar.

¡Oh, no! Actualmente no hay clientes para mostrar.

MSJAPI-12 Sin clientes verificados.

¡Oh, no! Actualmente no hay clientes listos para ser aprobados.

MSJAPI-13 Rechazar cliente.

¿Seguro que desea rechazar el cliente? Esta acción no puede deshacerse.

MSJAPI-14 Error al actualizar el estado del cliente cliente.

Ocurrió un error al actualizar el estado del cliente; inténtelo de nuevo.

MSJAPI-15 Vetar cliente.

¿Está seguro de que quiere vetar a este cliente?

MSJAPI-16 Cliente ya ha iniciado el refresco de llaves.

El proceso de refresco de llaves ya se ha iniciado.

MSJAPI-17 Cliente no ha iniciado el refresco de llaves.

No es posible terminar el refresco de llaves porque no ha iniciado el proceso de refresco de llaves.

MSJAPI-18 Error al iniciar el refresco de llaves.

¡Oh, no! Ocurrió un error al iniciar el refresco de llaves, por favor, inténtelo de nuevo más tarde.

MSJAPI-19 Error al terminar el refresco de llaves.

¡Oh, no! Ocurrió un error al terminar el refresco de llaves; por favor, inténtelo de nuevo más tarde.

MSJAPI-20 Advertencia de retokenización incompleta.

Aún existen tokens que no han sido actualizados, ¿seguro que desea terminar el refresco de llaves?

Estos tokens se volverán inservibles.

MSJAPI-21 Advertencia al eliminar un cliente.

¿Está seguro que desea eliminar su cuenta? Esta acción es irreversible.

6.1.6. Modelo de datos

6.1.7. Seguimiento de requerimientos de TOKENS

A continuación se encuentran los requerimientos que caen en la clasificación del SISTEMA TOKENIZADOR mencionados en la sección 4.1. Se indica, para cada requerimiento, si el requerimiento fue satisfecho o no y cómo o por qué no.

- **REQFPCI-01 VALIDACIÓN DE PRODUCTOS DE HARDWARE.** Como no se utilizan productos de hardware en el sistema tokenizador, este requerimiento se satisface por vacuidad.
- **REQFPCI-02 VALIDACIÓN DE PRODUCTOS DE SOFTWARE.** Ya que este requerimiento necesita de una auditoria por parte del FEDERAL INFORMATION PROCESSING STANDARD (FIPS), este requerimiento no es satisfecho.
- **REQFPCI-05 DETECCIÓN DE ANOMALÍAS.** El contador de malas acciones es el encargado de encontrar *anomalías* en el uso del sistema tokenizador; es aumentado cuando se reciben peticiones con TOKENS o PERSONAL ACCOUNT NUMBERS (PANs) inválidos, o cuando se intenta detokenizar un TOKEN que no está asociado al cliente que mandó la petición (véase RNAPI-12 CONTADOR DE MALAS ACCIONES). Por lo tanto, este requerimiento es satisfecho.
- **REQFPCI-07 GUÍA DE INSTALACIÓN.** La API tiene documentación en su sitio en línea que permite saber cómo comunicarse con ella y cuáles son los posibles errores; además, el código es libre y público (véase REQFAPI-23 MOSTRAR DOCUMENTACIÓN DEL SERVICIO TOKENIZADOR). Por lo tanto, este requerimiento es satisfecho.

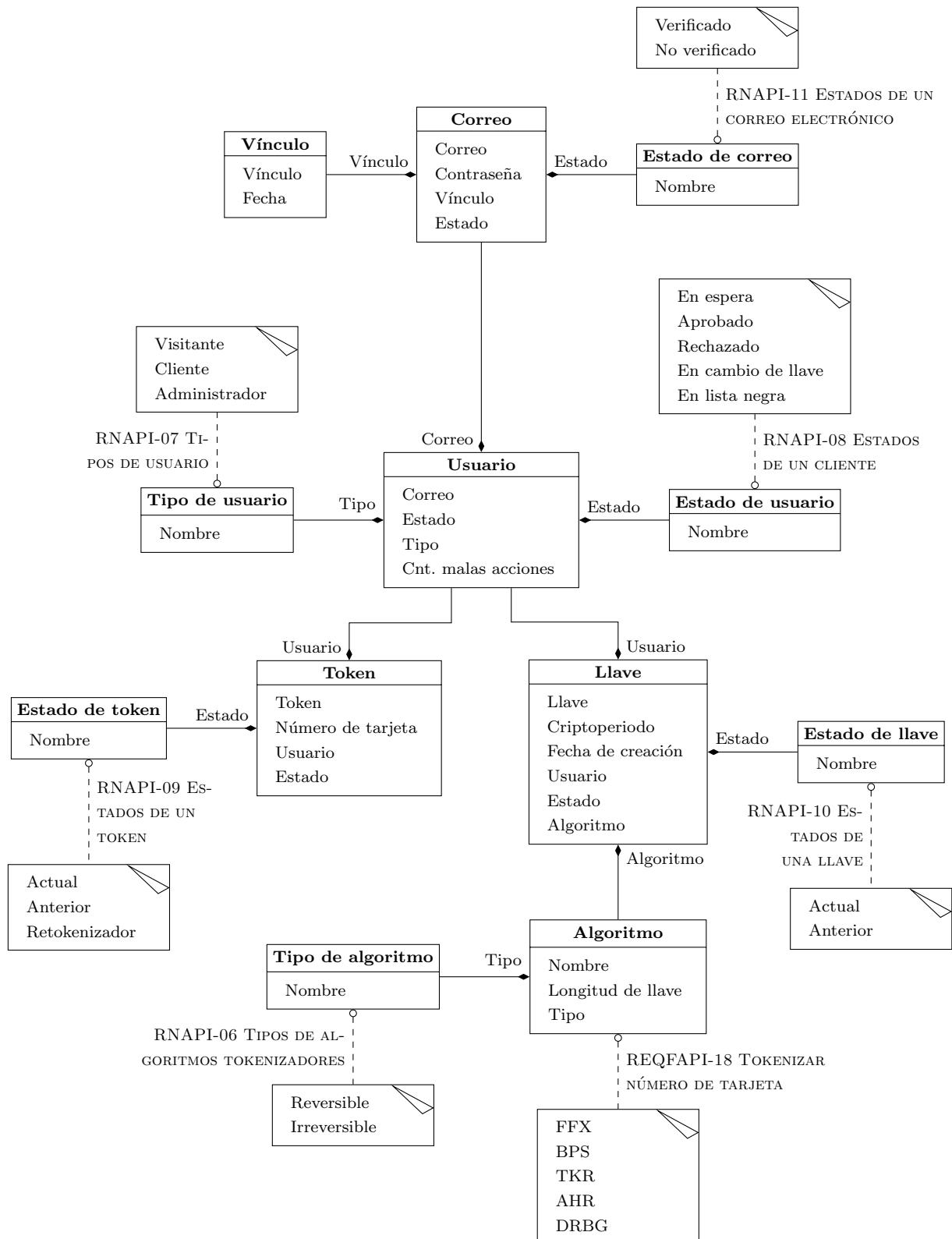


Figura 6.18: Modelo de datos de aplicación.

- **REQFPCI-09 ACCESO AL PROCESO DETOKENIZACIÓN.** Este requerimiento contiene varios subrequerimientos que se listan a continuación:

- **SUBREQFPCI-09/1 CONTROL DE PETICIONES.** Ya que la única manera de acceder al sistema tokenizador es mediante peticiones y en cada una se debe mandar un token de autenticación, este requerimiento es satisfecho (véase REQFAPI-18 TOKENIZAR NÚMERO DE TARJETA, REQFAPI-19 DETOKENIZAR UN TOKEN, REQFAPI-20 RETOKENIZAR UN TOKEN).
- **SUBREQFPCI-09/2 REGISTROS DE ACCESO.** Ya que el sistema no mantiene un registro de los eventos y las operaciones realizadas, este requerimiento no es satisfecho.
- **SUBREQFPCI-09/3 AUTENTICACIÓN MULTIFACTOR.** Debido al alcance (y presupuesto) del proyecto, no se cuenta con autenticación multifactor; por lo tanto, este requerimiento no es satisfecho.
- **SUBREQFPCI-09/4 ACCESOS A NIVEL DE SISTEMA.** Los clientes se identifican en cada petición ante el sistema mediante sus credenciales (usuario y contraseña); el sistema se identifica ante el cliente mediante TLS/SSL. Por lo tanto, este requerimiento es satisfecho.
- **SUBREQFPCI-09/5 ACCESOS ADMINISTRATIVOS.** Ya que no tenemos accesos administrativos desde fuera de la consola, este requerimiento es satisfecho por vacuidad. El usuario tipo administrador no tiene permisos para utilizar el programa tokenizador.

Como se puede observar, este requerimiento es satisfecho parcialmente, pues hay subrequerimientos con los que no se cumple.

- **REQFPCI-13 SOBRE EL MANEJO ADECUADO DE LLAVES.** Este requerimiento contiene varios subrequerimientos que se listan a continuación:

- **SUBREQFPCI-13/1 SOBRE EL CICLO DE VIDA.** Los criptoperiodos por defecto en el sistema son de 6 meses, lo que es un tiempo relativamente corto; sin embargo, el sistema no inicia por sí mismo un refresco de llaves, sino que es el cliente quien debe realizar el refresco (véase RNAPI-18 CRIPTOPERIODO DE UNA LLAVE). Por lo tanto, como no es responsabilidad del sistema, este requerimiento es satisfecho.
- **SUBREQFPCI-13/2 DESCRIPCIÓN DEL PERÍODO CRIPTOGRÁFICO ACTIVO.** Dependiendo de a quién se le brinde la atención, el servicio de tokenización puede ser muy variado, dependiendo de las necesidades de cada cliente. Ya que no se describe el periodo criptográfico, este requerimiento no es satisfecho.
- **SUBREQFPCI-13/3 SOBRE LA DESTRUCCIÓN DE LAS LLAVES.** El cliente puede, en cualquier momento hacer un refresco de llaves (destruir las nuevas y crear unas nuevas) sin tener que alterar el sistema; por lo tanto, el requerimiento es satisfecho.
- **SUBREQFPCI-13/4 EXPORTAR LLAVE EN CLARO PROHIBIDO.** El sistema no permite exportar llaves, así que este requerimiento es satisfecho.

- **SUBREQPCI-13/5 ENTROPIA DE GENERACIÓN DE LLAVES.** Las llaves son generadas mediante un DRBG que funciona con una función hash que tiene 256 bits de entropía; por lo tanto, este requerimiento es satisfecho.
- **SUBREQPCI-13/6 LLAVES DE USO ÚNICO.** Cada cliente tiene un juego de llaves; una para cada algoritmo; además, todas las llaves son únicas en la base de datos y no se utilizan para nada más. Por lo tanto este requerimiento es satisfecho.
- **REQFPCI-16 ALMACENAMIENTO DE TOKENS (CRIPTOGRÁFICOS).** Para los algoritmos reversibles implementados (FFX y BPS), el sistema no guarda los TOKENS generados; por lo tanto, este requerimiento es satisfecho.
- **REQFPCI-17 SEGURIDAD DE LA ADMINISTRACIÓN DE LLAVES (CRIPTOGRÁFICOS).** Dado el alcance del proyecto y que este requerimiento involucra auditorías o validaciones de organizaciones externas, no es satisfecho.
- **REQFPCI-23 INSTANCIAS ESTADÍSTICAMENTE INDEPENDIENTES (NO CRIPTOGRÁFICOS).** Ya que el sistema tiene una sola instancia de la base de datos, este requerimiento es satisfecho.
- **REQFPCI-26 SEGURIDAD DE LA ADMINISTRACIÓN DE LLAVES (NO CRIPTOGRÁFICOS).** Dado el alcance del proyecto y que este requerimiento involucra auditorías o validaciones de organizaciones externas, no es satisfecho.

6.2. Diseño

En esta sección se detallan los aspectos más importantes del diseño de la aplicación web que interactúa con el programa generador de tokens.

6.2.1. Diseño de la base de datos

6.2.2. Vista estática

6.2.3. Vista dinámica

En las figuras 6.22 y 6.23 se muestran, a modo de ejemplo, diagramas de secuencia de los casos de uso para registrar un cliente (CUAPI-03 REGISTRAR CLIENTE) y para tokenizar un número de tarjeta (CUAPI-12 TOKENIZAR UN NÚMERO DE TARJETA); todos los demás casos de uso mantienen una estructura análoga. En la figura 6.24 se muestra el comportamiento dinámico del sistema para las operaciones CREATE, READ, UPDATE AND DELETE (CRUD).

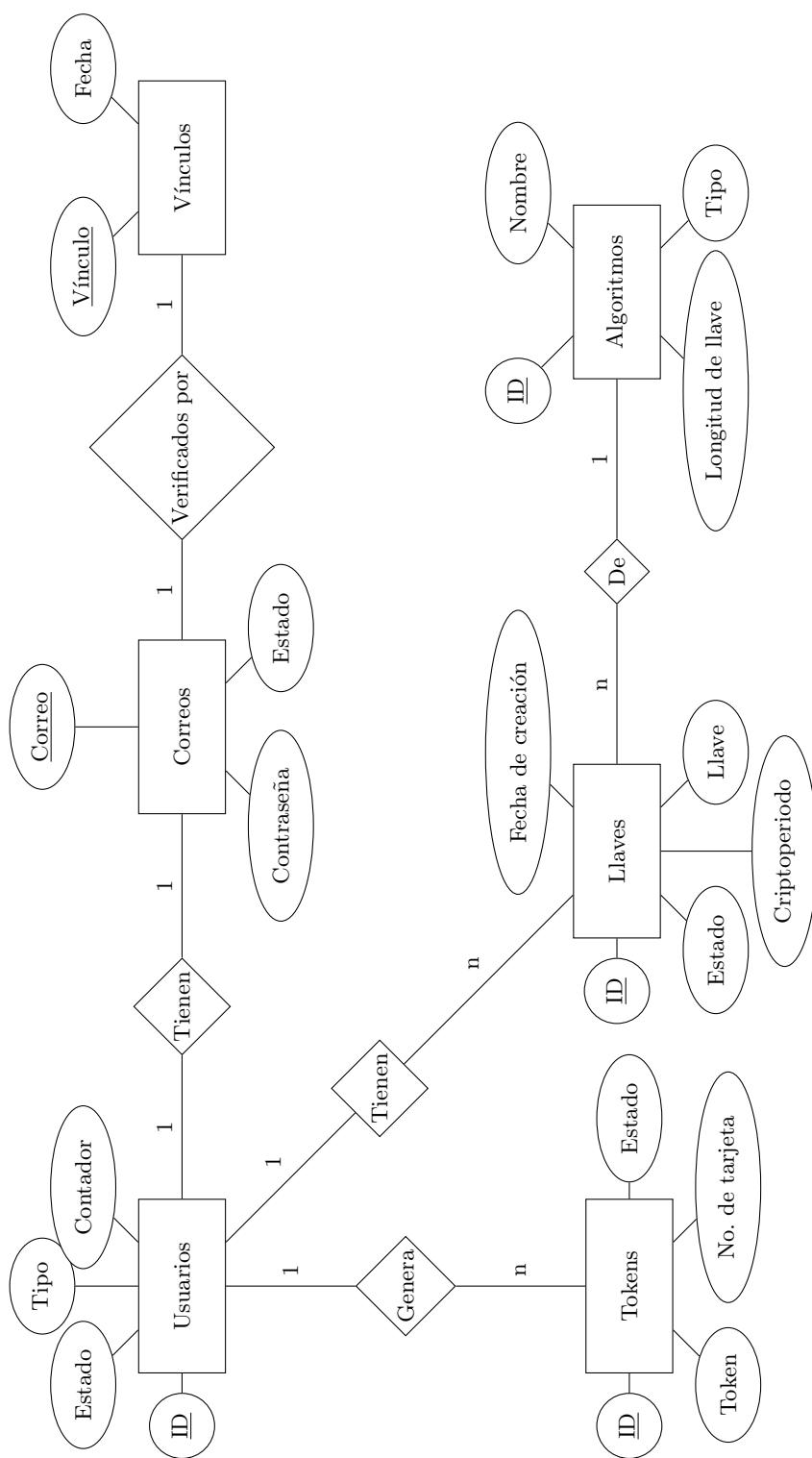


Figura 6.19: Diagrama Entidad-Relación de la base de datos de la aplicación.

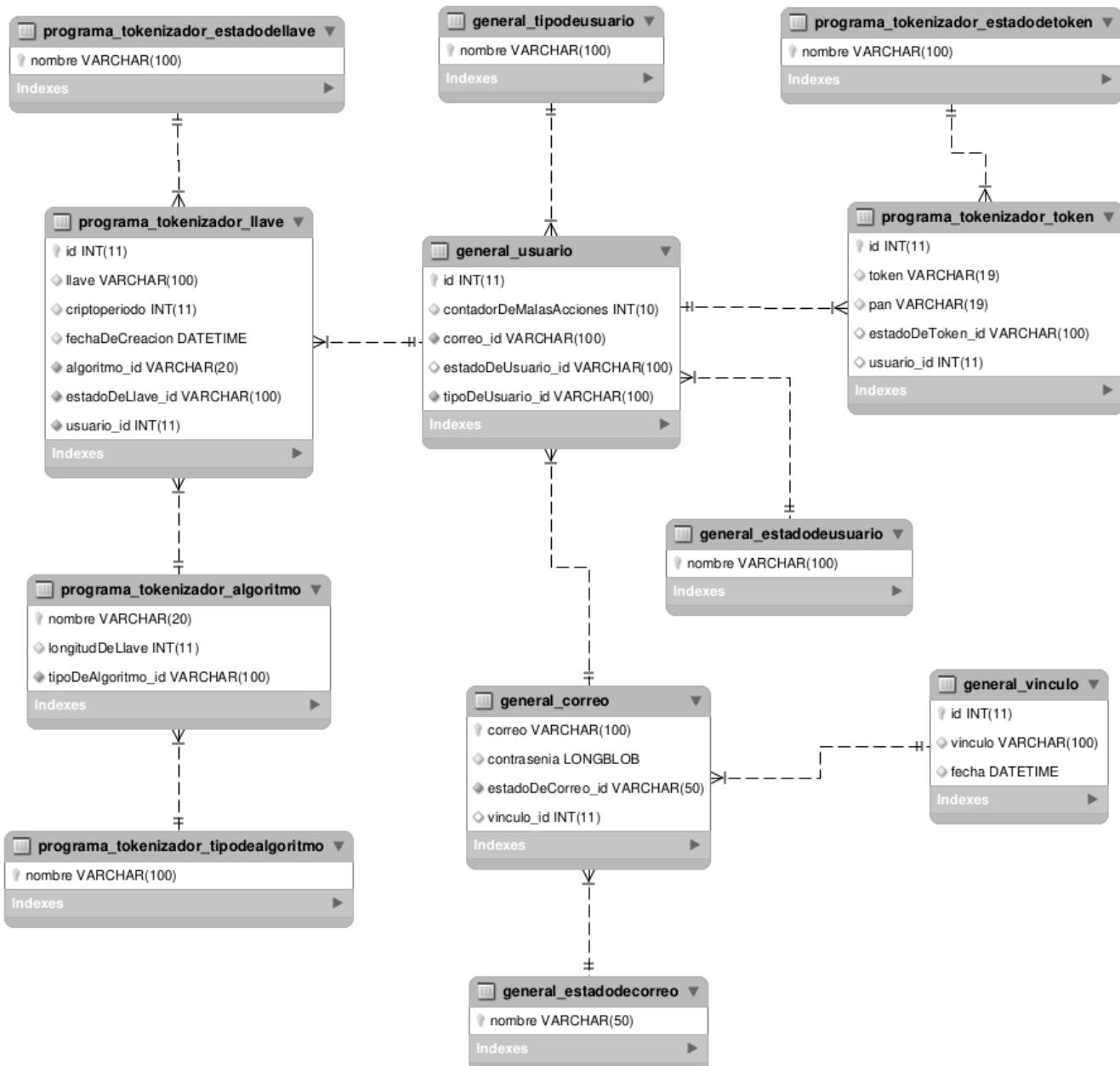


Figura 6.20: Diagrama relacional de la base de datos.

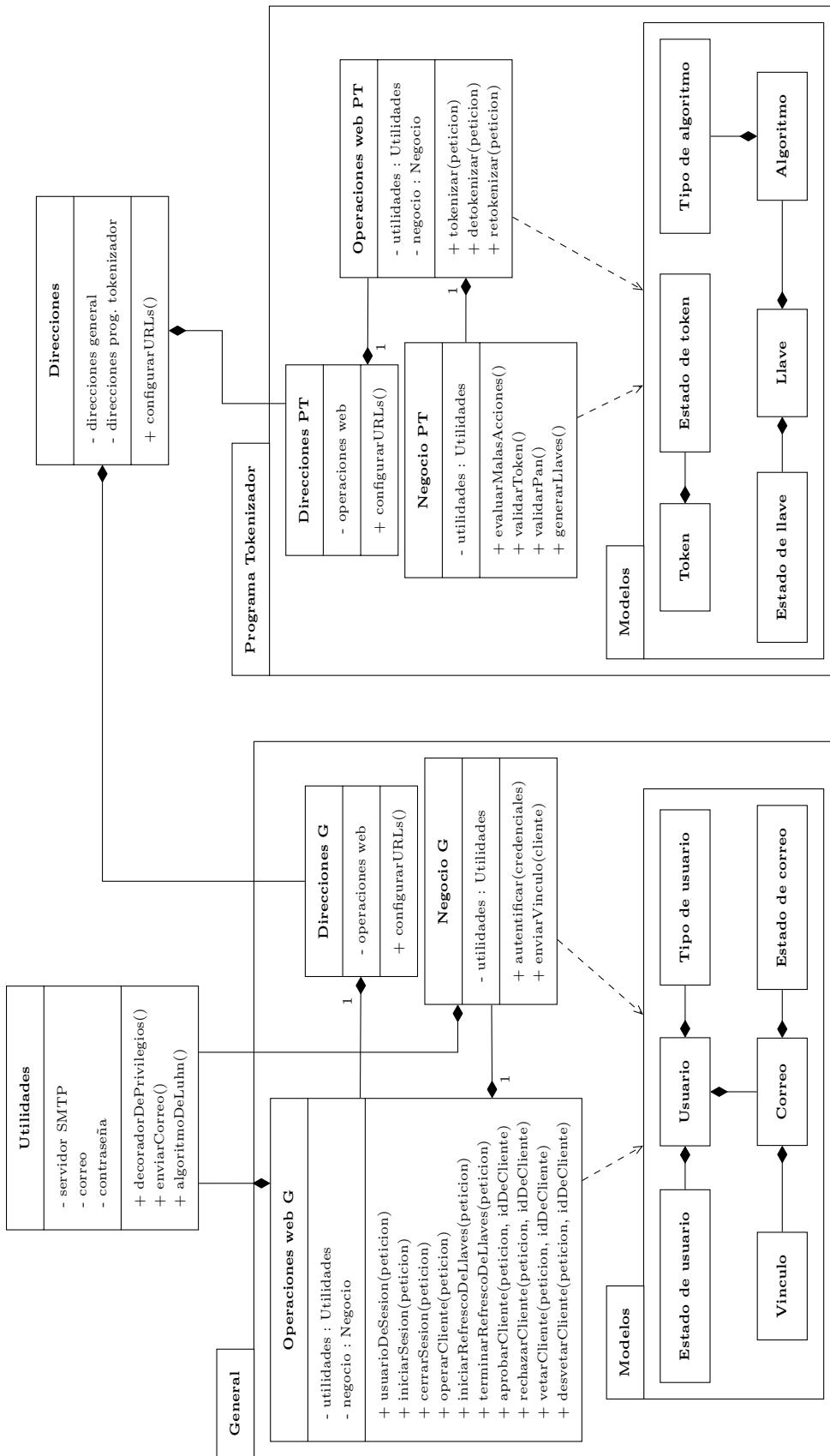


Figura 6.21: Diagrama de clases de aplicación web.

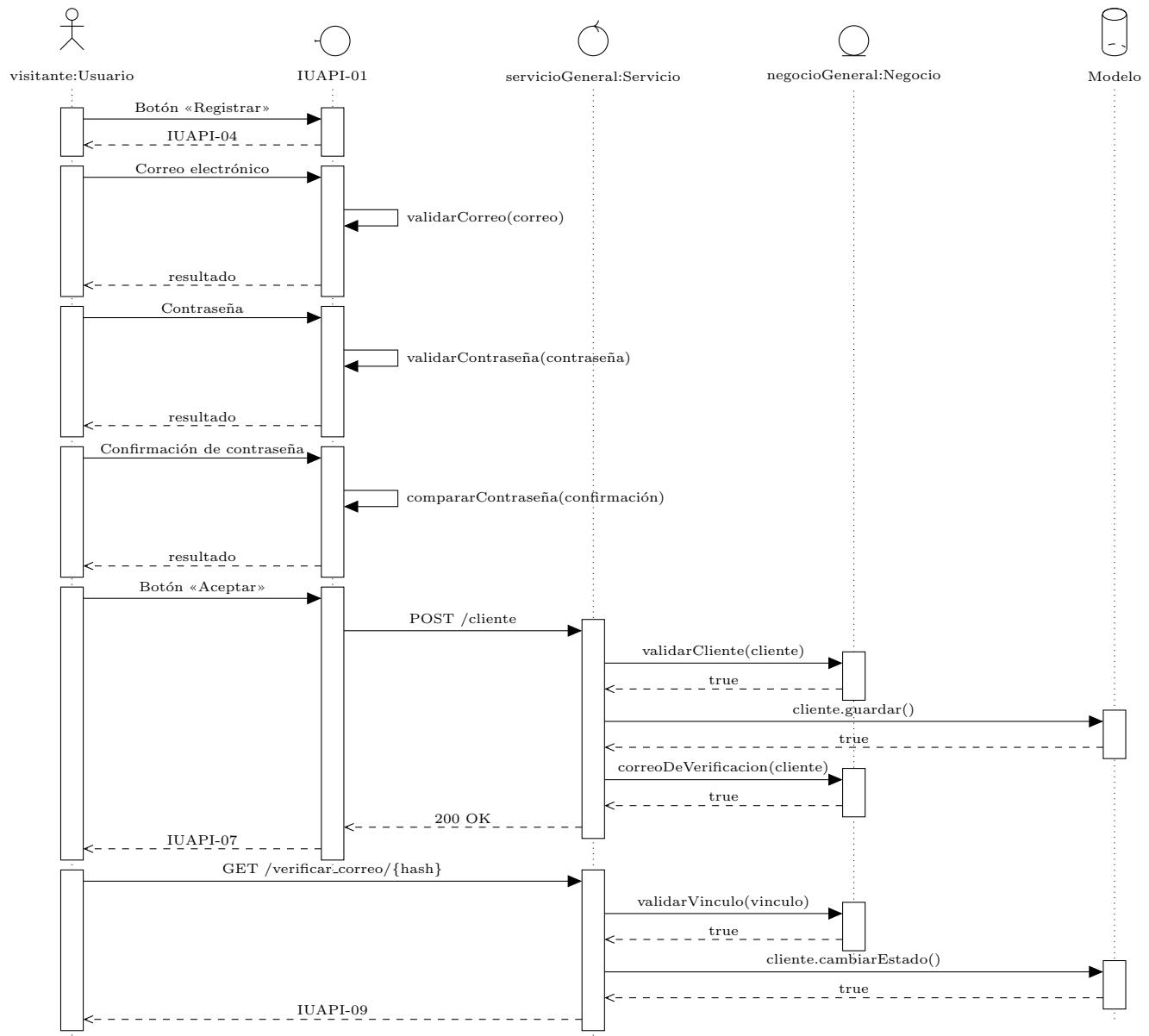


Figura 6.22: Diagrama de secuencia para registrar a un cliente.

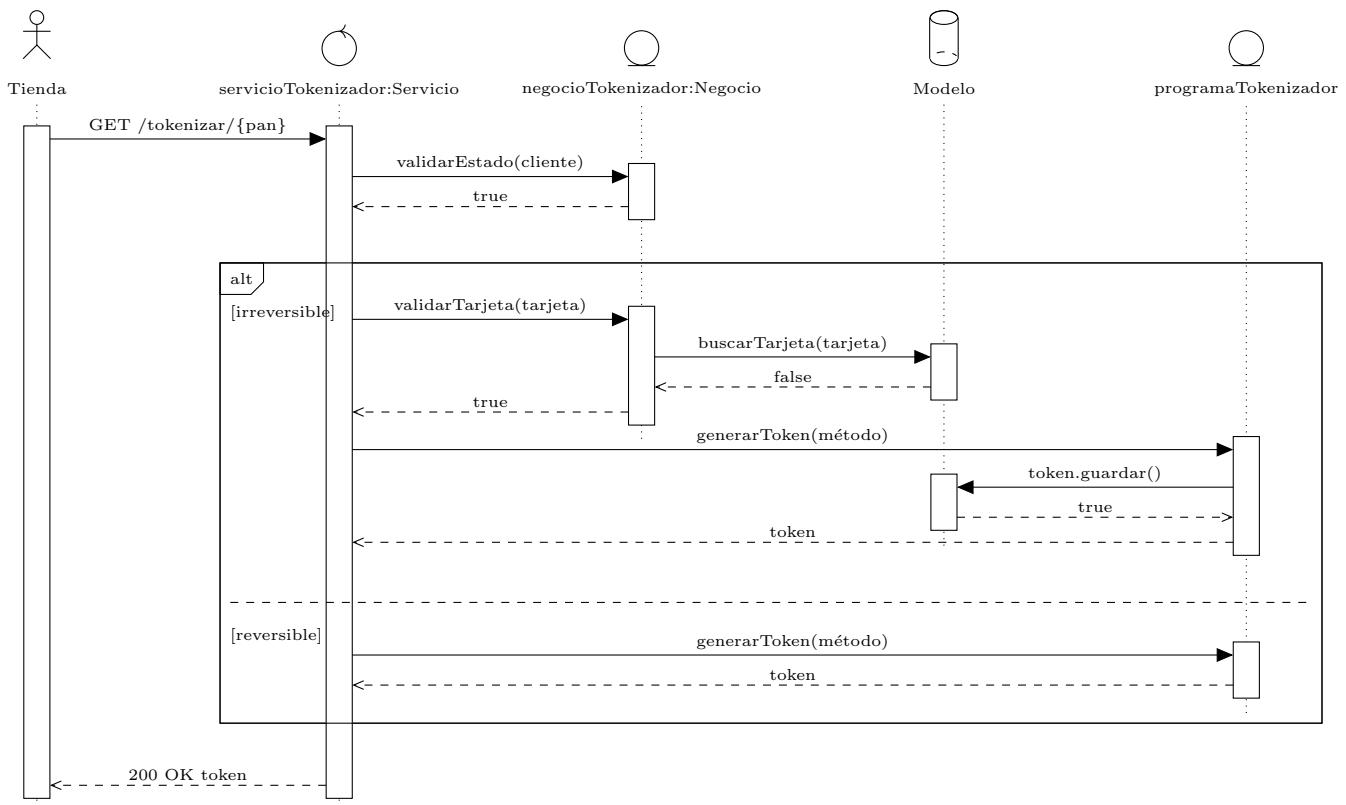


Figura 6.23: Diagrama de secuencia para operación de tokenización.

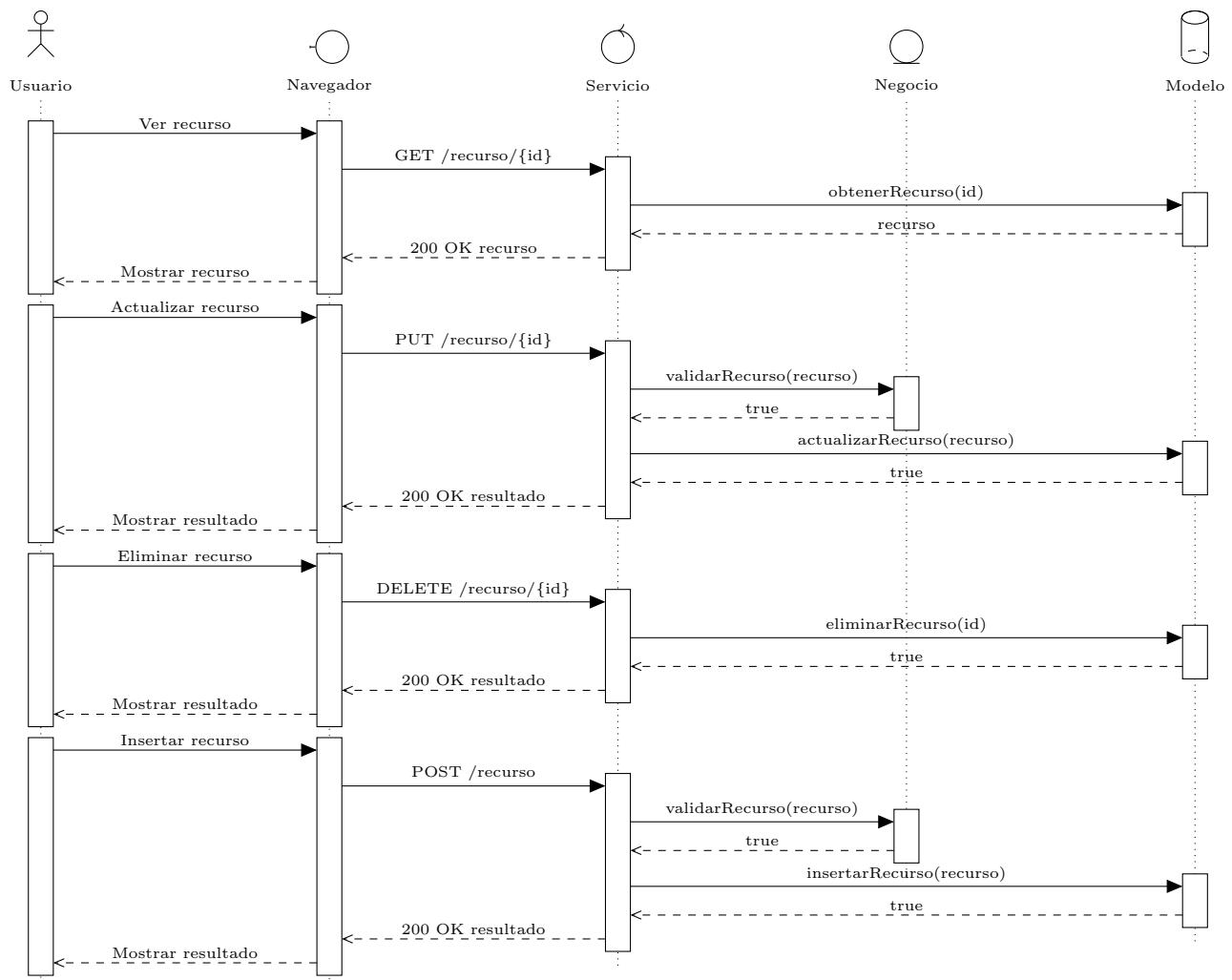


Figura 6.24: Diagrama de secuencia para operaciones genéricas.

Capítulo 7

Implementación de aplicación web

«From then on, when anything went wrong with a computer, we said it had bugs in it.»

GRACE HOPPER.

En este capítulo se presentan algunos de los aspectos más interesantes relacionados con la implementación de la aplicación descrita en el capítulo anterior. Primero se describen las tecnologías usadas, tanto de distribución como de desarrollo; después, se presentan algunos ejemplos del código, los suficientes como para poder exemplificar cómo es todo el programa; por último se presentan mediciones de tiempo de respuesta del servidor frente a peticiones de tokenización y detokenización.

7.1. Tecnologías

Este módulo se encuentra programado en *Python* y *Javascript*.

7.1.1. Dependencias de distribución

La dependencia DEPPT-02 GESTOR DE BASE DE DATOS: MARIADB v10.1, del programa tokenizador, también se ocupa en este módulo.

DEPAPI-01 Framework web en servidor: Django v2.1.2.

Licencia: Propia

Página oficial: <https://www.djangoproject.com>

DEPAPI-02 Framework web en cliente: Angular v1.7.4.

Licencia: MIT

Página oficial: <https://angularjs.org>

DEPAPI-03 Componentes gráficos: Angular materialv1.1.10.

Licencia: MIT

Página oficial: <https://material.angularjs.org>

DEPAPI-04 Ejecutor de tareas: Gruntv1.0.3.

Licencia: MIT

Página oficial: <https://gruntjs.com/>

DEPAPI-05 Extensión de CSS: Sassv1.14.

Licencia: MIT

Página oficial: <http://sass-lang.com/>

7.1.2. Dependencias de desarrollo

Las siguientes dependencias son heredadas desde el módulo anterior:

- DEPPT-05 CONTROL DE VERSIONES: GIT v2.17
-
- DEPPT-07 HERRAMIENTA DE INTEGRACIÓN CONTINUA: TRAVIS

DEPAPI-06 Gestor de dependencias en el cliente: NPMv6.4.1.

Licencia: MIT

Página oficial: <https://www.npmjs.com/>

DEPAPI-07 Gestor de dependencias en el servidor: PIPv18.

Licencia: MIT

Página oficial: <https://www.npmjs.com/>

7.2. Aspectos relevantes de la implementación

En esta sección se muestran, a modo de ejemplo, algunos de los aspectos más relevantes de la implementación del programa descrito en el capítulo anterior.

7.2.1. Ejemplos del programa servidor

El programa está dividido en dos grandes módulos: lo referente al programa tokenizador, y lo referente a las operaciones web (diagrama de clases 6.21). Cada módulo tiene una sección de direcciones, una de operaciones de negocio, una de operaciones web y una de clases de modelos; existe también una sección de utilidades generales. A continuación se muestran ejemplos de cada una de estas secciones.

El código fuente 22 muestra una operación web del módulo general: la petición de paginador para los clientes **aprobados**. Existen funciones similares para los clientes en estado de **lista negra** y los clientes **en espera**. Estas funciones se utilizan para armar e interactuar con los paginadores de la interfaz de un administrador (pantalla IUAPI-06 PÁGINA DE ADMINISTRACIÓN GENERAL).

En el código fuente 23 se muestra una operación de negocio del módulo general: la autentificación de los usuarios. Esta función se utiliza al momento de un inicio de sesión. Este código representa el cumplimiento del requerimiento REQNFAPI-02 SOBRE EL ALMACENAMIENTO DE CONTRASEÑAS, ya que para realizar el proceso de autentificación se calcula el hash de la entrada del usuario y se compara contra lo que hay guardado en la base de datos.

```
474 @utilidades.privilegiosRequeridos('administrador')
475 def obtenerClientesAprobados(peticion, pagina, limite):
476     """Función de paginador para clientes aprobados.
477
478     Regresa el rango solicitados de clientes aprobados.
479     El objeto Q es para hacer consultas con OR: «filter» y
480     «get» funcionan con AND.
481     https://docs.djangoproject.com/en/2.1/topics/db/queries/
482     #complex-lookups-with-q-objects"""
483     todos = Usuario.objects.filter(
484         django.db.models.Q(tipoDeUsuario = TipoDeUsuario.objects.get(
485             nombre = 'cliente')),
486         django.db.models.Q(estadoDeUsuario = EstadoDeUsuario.objects.get(
487             nombre = 'aprobado')) |
488         django.db.models.Q(estadoDeUsuario = EstadoDeUsuario.objects.get(
489             nombre = 'en cambio de llaves'))).order_by('correo')
490     return utilidades.respuestaJSON(
491         todos[(pagina - 1) * limite : pagina * limite])
```

Código fuente 22: Paginador de clientes aprobados.

SISTEMA_TOKENIZADOR/GENERAL/FUNCIONES_WEB.PY

```
19 def autenticar(usuarioEnPeticion):
20     """Valida el usuario dado (correo y contraseña).
21
22     En caso de no existir, regresa None; en caso correcto,
23     regresa el objeto del usuario."""
24     try:
25         return Usuario.objects.get(
26             correo = Correo.objects.get(
27                 correo = usuarioEnPeticion['correo'],
28                 contrasenia = hashlib.sha256(
29                     usuarioEnPeticion['contrasenia'].encode('UTF-8')).digest()))
30     except (Usuario.DoesNotExist, Correo.DoesNotExist):
31         return None
```

Código fuente 23: Operación de autentificación de usuarios.

SISTEMA_TOKENIZADOR/GENERAL/NEGOCIO.PY

Como ejemplo de operación web del módulo tokenizador, los códigos fuente 24 y 25 muestran la operación de tokenización. Esta función representa casi la totalidad del caso de uso CUAPI-12 TOKENIZAR UN NÚMERO DE TARJETA. En esta función se puede apreciar mejor que en los códigos anteriores la interacción entre las distintas capas: las validaciones se hacen en la capa de negocio (línea 128); el módulo de configuraciones almacena constantes importantes (línea 144), en este caso, se trata de la ruta del ejecutable al programa generador de tokens. Todos los códigos de error y mensajes que regresa esta función son los descritos en la página de documentación ([HTTPS://RICARDO-QUEZADA.159.56.43.6/LOVELACE/DOCUMENTACION](https://RICARDO-QUEZADA.159.56.43.6/LOVELACE/DOCUMENTACION)).

```
107 def tokenizar (peticion):
108     """Ejecuta la operación de tokenización y regresa el token asignado.
109
110     Si ya se había tokenizado ese PAN, regresa el token creado. Si se encuentra
111     en cambio de llaves y tiene el actual y el viejo/retokenizado, se regresa
112     el actual. Si está en cambio de llaves y solo tiene el viejo, se regresa
113     un error, pidiendo que retokenice."""
114
115     cliente = autentificar(peticion)
116
117     if isinstance(cliente, django.http.HttpResponse):
118         return cliente
119
120     objetoDePeticion = json.loads(peticion.body)
121     try:
122         pan = objetoDePeticion['pan']
123         metodo = objetoDePeticion['metodo'].upper()
124     except:
125         return django.http.HttpResponse("Parámetros incompletos o incorrectos",
126                                         status = 403)
127
128     if negocio.validarPan(pan) == 0:
129         return django.http.HttpResponse("El PAN recibido es inválido.",
130                                         status = 400)
131
132     tipoAlgoritmo = Algoritmo.objects.get(nombre = metodo).tipoDeAlgoritmo_id
133
134     ## Si el algoritmo es reversible o es irreversible y el PAN no está registrado
135     ## en la base de datos, crear un nuevo token y regresarlo.
136     if tipoAlgoritmo == 'reversible' or (tipoAlgoritmo == 'irreversible' and \
137         (negocio.verificarUnicidadDePAN(pan, cliente.id) == 1)):
138         llave = Llave.objects.get(
139             algoritmo_id = Algoritmo.objects.get(nombre = metodo),
140             usuario_id = cliente.id,
141             estadoDeLlave_id = EstadoDeLlave.objects.get(nombre = 'actual'))
142
143         resultado = subprocess.run([
144             configuraciones.EJECUTABLE_TOKENIZADOR,
145             "-e",
146             metodo,
147             pan,
148             llave.llave,
149             str(cliente.id)], stdout = subprocess.PIPE)
150
151     return django.http.HttpResponse(resultado.stdout, status = 200)
```

Código fuente 24: Operación de tokenización (primera parte).

SISTEMA_TOKENIZADOR/PROGRAMA_TOKENIZADOR/FUNCIONES_WEB.PY

```
153     ## Si el PAN especificado ya está asociado al cliente, regresar el token si
154     ## el estado del cliente es aprobado.
155     if cliente.estadoDeUsuario.nombre == 'aprobado':
156         return django.http.HttpResponse(
157             Token.objects.get(
158                 usuario_id = cliente.id,
159                 pan = pan).token,
160             status = 403)
161
162     ## Si está en cambio de llaves, puede tener uno o dos tokens asociados a ese
163     ## PAN.
164     tokens = Token.objects.filter(
165         usuario_id = cliente.id,
166         pan = pan)
167
168     ## Si solo tiene uno y es el actual, regresar ese token; si el token tiene
169     ## otro estado, debe realizar la operación de retokenización para obtener
170     ## la nueva versión.
171     if len(tokens) == 1:
172         if tokens[0].estadoDeToken.nombre == 'actual':
173             return django.http.HttpResponse(tokens[0].token, status = 403)
174         else:
175             return django.http.HttpResponse(
176                 'Ya existe un token asociado, utilice la función de retokenización',
177                 status = 400)
178
179     ## Si tiene dos, uno es el actual y otro es el viejo o retokenizado; regresar
180     ## el actual.
181     if tokens[0].estadoDeToken.nombre == 'actual':
182         return django.http.HttpResponse(tokens[0].token, status = 403)
183     else:
184         return django.http.HttpResponse(tokens[1].token, status = 403)
```

Código fuente 25: Operación de tokenización (segunda parte).

SISTEMA_TOKENIZADOR/PROGRAMA_TOKENIZADOR/FUNCIONES_WEB.PY

En el código fuente 26 se muestra un ejemplo de función de negocio de módulo generador de tokens. Se trata de la gestión del contador de malas acciones (regla de negocio RNAPI-12 CONTADOR DE MALAS ACCIONES). Al inicio del código se muestra también cómo se gestionan las constantes de incrementos y límites. De esta forma se establecen en el código todas las constantes definidas en las reglas de negocio (esto solamente debe de ir en los archivos de capa de negocio).

```
23 LIMITE_MALAS_ACCIONES = 10
24 INCREMENTO_TOKEN_INVALIDO = 1
25 INCREMENTO_TOKEN_INEXISTENTE = 3
26
27
28 def aumentarContadorDeMalasAcciones (cliente, incremento):
29     """Incrementa el contador de malas acciones.
30
31     Incrementa el contador de malas acciones del cliente dado. Después, verifica
32     que el contador no haya sobre pasado el límite de males acciones; de ser así,
33     cambia su estado a «en lista negra»."""
34
35 cliente.contadorDeMalasAcciones = cliente.contadorDeMalasAcciones + incremento
36 cliente.save()
37
38 if cliente.contadorDeMalasAcciones > LIMITE_MALAS_ACCIONES:
39     cliente.estadoDeUsuario = \
40         general.models.estado_de_usuario.EstadoDeUsuario.objects.get(
41             nombre = 'en lista negra')
42     cliente.save()
```

Código fuente 26: Gestión de contador de malas acciones.

SISTEMA_TOKENIZADOR/PROGRAMA_TOKENIZADOR/NEGOCIO.PY

La realización de las capas de modelos de ambos módulos es a través de la APPLICATION PROGRAM INTERFACE (API) de *Django* (descrita de modo general en [53]). En el código fuente 27 se muestra la clase de modelo de un token. Django toma esta descripción para producir las funciones básicas de un CREATE, READ, UPDATE AND DELETE (CRUD). Ya se ha visto en los códigos anteriores cómo es la interacción con esta API (métodos como `save`, `filter`, `get` o `delete`).

Por último, como ejemplo de la capa general de utilidades, en el código 28 se muestra una de las funciones más usadas en las implementaciones de las funciones web: la validación de los privilegios del usuario en sesión. Por ejemplo, esta función se utiliza en la línea 33 del código fuente 22 para validar que solamente un usuario de tipo **administrador** pueda acceder al paginador de los clientes aprobados. La función `privilegiosRequeridos` funciona como una fábrica de decoradores: toma como argumento el nombre del tipo de usuario y regresa una función decorador para ese tipo de usuario. Los decoradores en *python* son funciones que toman como argumentos a la función decorada: intercalan el nuevo código (en este caso la validación de las credenciales) con la llamada a la función de entrada [54].

Con la función de validación de privilegios se ocupan dos patrones de diseño: una fábrica y un decorador. Aunque su realización es distinta de lo que se esperaría en un lenguaje completamente orientado a

```

1 """
2   Modelo de registro PAN-token para programa tokenizador,
3   Aplicación web de sistema tokenizador.
4   Proyecto Lovelace.
5 """
6
7 import django
8 from .estado_de_token import EstadoDeToken
9
10
11 def estadoPorDefecto():
12     return EstadoDeToken.objects.get(nombre = 'actual')
13
14
15 class Token(django.db.models.Model):
16     """Relación entre PAN-token."""
17
18     token = django.db.models.CharField(
19         verbose_name = 'Token',
20         max_length = 19)
21
22     pan = django.db.models.CharField(
23         verbose_name = 'Número de tarjeta',
24         max_length = 19)
25
26     usuario = django.db.models.ForeignKey(
27         'general.Usuario',
28         django.db.models.PROTECT,
29         verbose_name = 'Usuario dueño del token',
30         default = None,
31         null = True)
32
33     estadoDeToken = django.db.models.ForeignKey(
34         'EstadoDeToken',
35         django.db.models.PROTECT,
36         verbose_name = 'Estado del token',
37         default = estadoPorDefecto,
38         null = True)
39
40     class Meta:
41         unique_together = ('usuario', 'token')
42
43     def __str__(self):
44         """Representación en cadena de un tipo de usuario."""
45         return str(self.id) + ' - ' + self.token + ' - ' + self.token

```

Código fuente 27: Modelo de token.

SISTEMA_TOKENIZADOR/PROGRAMA_TOKENIZADOR/MODELS/TOKEN.PY

objetos y, por lo tanto, de la descrita en [55], la funcionalidad obtenida en un último término corresponde a la obtenida por los patrones de diseño originales. La fábrica permite la creación de objetos (en este caso, métodos) con una estructura determinada en tiempo de ejecución; aquí, la estructura se encuentra representada por el tipo de usuario al que se le desean dar privilegios. Los decoradores permiten extender o alterar la funcionalidad de un método: el decorador agrega la validación de privilegios justo antes de ejecutar la propia función decorada.

7.2.2. Ejemplos del programa cliente

En el código fuente 29 se muestra el inicio del HYPERTEXT MARKUP LANGUAGE (HTML) de la ventana para operar a un cliente (dependiendo del contexto, se puede tratar de la interfaz IUAPI-04 FORMULARIO DE REGISTRO DE USUARIO o de IUAPI-11 FORMULARIO DE EDICIÓN DE DATOS DE UN CLIENTE). Como se puede observar al inicio del archivo, hay reglas que permiten modificar el tamaño de las ventanas dependiendo del tamaño de la pantalla del usuario.

Cada HTML tiene un controlador de *javascript* asociado. En el caso de la ventana de operación de un cliente (código 29), un fragmento de su controlador se muestra en el código fuente 31. *Angular* sigue un patrón de inyección de dependencias [56] para la construcción de controladores: después del nombre dado al controlador, recibe un arreglo con cadenas que representan los nombres de las dependencias que el inyector tiene que buscar para construir la función (el último elemento del arreglo recibido) [57].

Los dos códigos anteriores, la ventana y su controlador, dejan ver cómo es la gestión de los mensajes y las expresiones regulares en el código del cliente. En los archivos HTML hay marcadores con formato `<!-- @@ -->` (línea 42 del código 29); en los archivos *javascript* hay marcadores con formato `@@()` (línea 26 del código 31). Estos marcadores son sustituidos en tiempo de compilación por un componente de *Grunt* (*includereplace*): el marcador indica un archivo de texto plano en donde se encuentra definido el mensaje o la regla de negocios en cuestión. Estos archivos se producen en una etapa de análisis y son los mismos que utiliza este documento en la sección de reglas de negocio (6.1.1) o la sección del catálogo de mensajes (6.1.5).

El punto anterior busca satisfacer una buena práctica de programación: las definiciones redundantes son malas; la información debe de estar en un solo lugar y de ahí debe de ser usada por los códigos dependientes. Por ejemplo, en un supuesto de que se quiera cambiar el contenido de un mensaje, o modificar una expresión regular, solamente hay que modificar el archivo de texto correspondiente y ambas dependencias, programa y documentación, se adecuarán al cambio. Este principio es conocido como DON'T REPEAT YOURSELF (DRY) y es uno de los principios de diseño de las aplicaciones hechas en *Django* [58].

La comunicación con el programa servidor se define en una capa separada de los propios controladores. En el código fuente ?? se muestra un fragmento de esta capa. Este es un modo de separar las direcciones y fuentes de datos que se obtienen del servidor, del propio funcionamiento de cada controlador.

```

33 def privilegiosRequeridos (tipoDeUsuario):
34     """Fábrica de decoradores para operaciones web con privilegios.
35
36     Permite decorar las funciones web (todas las definiciones de funciones
37     que hay en cualquier funciones_web.py) para restringir el uso de esa función
38     a cierto tipo de usuario. El argumento recibido es una cadena con el nombre
39     del tipo de usuario que debe poder acceder a la función.
40
41     Si no hay ningún usuario en sesión o el usuario de sesión no tiene los
42     privilegios necesarios, se regresa un 304 para obligar al cliente a ir
43     a la pantalla de inicio de sesión.
44
45     Las fábricas de decoradores se ven, en el código cliente, igual que
46     un decorador normal. En su papel de fábrica, debe de regresar un decorador.
47     Las fábricas de decoradores se utilizan para que las funciones decoradoras
48     puedan recibir parámetros (en este caso el tipo de usuario de los
49     privilegios). Cuando el decorador no recibe parámetros se puede utilizar
50     sin la fábrica.
51
52     Definitivamente, mi función favorita."""
53
54 def decorador (funcion):
55
56     @functools.wraps(funcion)
57     def envolturaDePrivilegios(peticion, *argumentos, **argumentosEnDiccionario):
58
59         if 'usuario' not in peticion.session:
60             return django.http.HttpResponseRedirect('/?siguiente=' + peticion.path)
61
62         else:
63             usuario = None
64             for objetoDeserializado \
65                 in django.core.serializers.deserialize("json",
66                     peticion.session['usuario']):
67                 usuario = objetoDeserializado
68                 break
69             if str(usuario.object.tipoDeUsuario) != tipoDeUsuario:
70                 return django.http.HttpResponseRedirect('/?siguiente=' + peticion.path)
71
72         else:
73             return funcion(peticion, *argumentos, **argumentosEnDiccionario)
74
75     # Retorno de función decoradora, decorador.
76     return envolturaDePrivilegios
77
78 # Retorno de función fábrica, privilegiosRequeridos.
79 return decorador

```

Código fuente 28: Fábrica de decoradores para validación de privilegios.

SISTEMA_TOKENIZADOR/UTILIDADES.PY

```
1 <!--
2   Formulario de actualización de clientes,
3   Aplicación web de sistema tokenizador.
4   Proyecto Lovelace.
5 -->
6
7 <md-dialog
8   flex="20"
9   flex-lg="30"
10  flex-md="40"
11  flex-sm="60"
12  flex-xs="80">
13  <md-dialog-content>
14    <md-toolbar class="md-primary">
15      <div class="md-toolbar-tools">
16        <h1>{{tituloOperacion}}</h1>
17      </div>
18    </md-toolbar>
19    <div
20      layout="row"
21      layout-align="center center">
22      <form
23        name="formularioOperarCliente"
24        id="formularioOperarCliente"
25        novalidate
26        flex="90"
27        ng-submit="operar(fevent)">
28        <md-input-container>
29          <label>Correo electrónico</label>
30          <input
31            name="correo"
32            ng-model="cliente.correo"
33            ng-value="usuario.fields.correo"
34            md-no-asterisk
35            required
36            md-autofocus
37            ng-pattern="erCorreo"
38            autocomplete="off"
39            spellcheck="false">
40          <div ng-messages="formularioOperarCliente.correo.$error">
41            <div ng-message="pattern">
42              <!-- @@include('mensajes/correo_incorrecto.txt') -->
43            </div>
44          </div>
45        </md-input-container>
```

Código fuente 29: Ventana emergente para operaciones de un cliente.

SISTEMA_TOKENIZADOR/ARCHIVOS_WEB/HTML/VENTANAS/OPERAR_CLIENTE.VENTANA.HTML

```

1  /*
2   * Controlador de formulario de inicio de sesión,
3   * Aplicación web de sistema tokenizador.
4   * Proyecto Lovelace.
5   */
6
7 sistemaTokenizador.controller('controladorFormularioOperarCliente', [
8   '$scope',
9   '$location',
10  '$mdDialog',
11  'api',
12  'tituloOperacion',
13  'operacion',
14  function (
15    $scope,
16    $location,
17    $mdDialog,
18    api,
19    tituloOperacion,
20    operacion
21  )
22  {
23   /* Datos públicos. *****/
24
25   $scope.cliente = {};
26   $scope.erCorreo = @@include('expresiones_regulares/correo.txt');
27   $scope.erContrasenia = @@include('expresiones_regulares/contrasenia.txt');
28   $scope.error = false;
29
30   $scope.tituloOperacion = tituloOperacion;
31   $scope.operacion = operacion;
32   mensaje = 'Para poder iniciar sesión es necesario que '
33     + 'verifique su cuenta accediendo al vínculo enviado a su '
34     + 'correo electrónico (revise su carpeta de spam).'
35
36   api.obtenerUsuarioDeSession().then(function (respuesta) {
37     if (respuesta.data != '') {
38       $scope.usuario = respuesta.data[0];
39     }
40   });

```

Código fuente 30: Controlador de ventana emergente para operaciones de un cliente.

SISTEMA_TOKENIZADOR/ARCHIVOS_WEB/JS/CONTROLADORES/SECUNDARIOS/OPERAR_CLIENTE.

CONTROLADOR.JS

```
1  /*
2   * Definición de comunicación con backend,
3   * Aplicación web de sistema tokenizador.
4   * Proyecto Lovelace.
5   */
6
7 sistemaTokenizador.factory('api', [
8   '$http',
9   function (
10    $http
11  )
12  {
13    var API = {};
14    var RUTA_BASE = '/api/general';
15
16    /* Operaciones de sesión. *****/
17
18    API.obtenerUsuarioDeSesion = function () {
19      return $http.get(RUTA_BASE + '/usuario_de_sesion');
20    };
21
22    API.iniciarSesion = function(usuario) {
23      return $http.post(RUTA_BASE + '/iniciar_sesion', usuario);
24    };
25
26    API.cerrarSession = function () {
27      return $http.get(RUTA_BASE + '/cerrar_sesion');
28    };
29
30    /* Operaciones de clientes. *****/
31
32    API.registrarCliente = function (cliente) {
33      return $http.post(RUTA_BASE + '/operar_cliente', cliente);
34    };
35
36    API.actualizarCliente = function (cliente) {
37      return $http.put(RUTA_BASE + '/operar_cliente', cliente);
38    };
39
40    API.eliminarCliente = function () {
41      return $http.delete(RUTA_BASE + '/operar_cliente');
42    };
43
44    API.iniciarRefrescoDeLlaves = function () {
45      return $http.post(RUTA_BASE + '/iniciar_refresco_de_llaves');
46    };

```

Código fuente 31: Definición de comunicación con programa servidor.

SISTEMA_TOKENIZADOR/ARCHIVOS_WEB/JS/SERVICIOS/API.SERVICIO.JS

	Tokenización	Detokenización
FFX	206.526 s	208.204 s
BPS	219.036 s	219.448 s
TKR	824.969 s	103.854 s
AHR	436.053 s	132.297 s
DRBG	437.406 s	124.064 s

Tabla 7.1: Comparación de tiempos.

7.3. Resultados

7.3.1. Desempeño de la API

Con la finalidad de probar el desempeño de la APPLICATION PROGRAM INTERFACE (API) desarrollada, se hizo la medición del los tiempo de respuesta de las peticiones de operaciones de tokenización y detokenización con los 5 métodos disponibles.

En la tabla 7.1 se pueden observar los tiempos obtenidos de las pruebas de desempeño de la API, los cuales se obtuvieron al realizar 10K peticiones de operaciones de tokenización y detokenización con cada método posible.

Con los datos de la tabla 7.1 se generó la gráfica de la figura 7.1 donde se puede ver de forma más clara la diferencia en los tiempos.

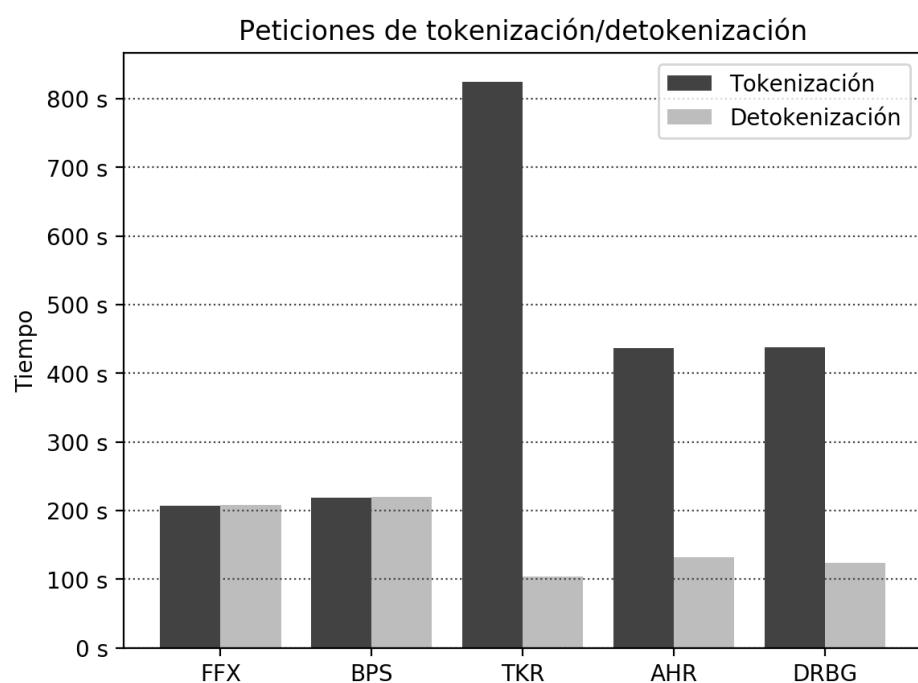


Figura 7.1: Comparación de tiempos de operación.

Parte III

Caso de prueba: tienda de libros en línea

«She believed in nothing; only her skepticism kept her from being an atheist.»

JEAN-PAUL SARTRE.

Capítulo 8

Análisis y diseño de tienda en línea

«Lolita, luz de mi vida, fuego de mis entrañas. Pecado mío, alma mía. Lo-li-ta: la punta de la lengua emprende un viaje de tres pasos paladar abajo hasta apoyarse, en el tercero, en el borde de los dientes. Lo. Li. Ta.»

Lolita, VLADIMIR NABOKOV.

En este capítulo se abordan los temas referentes al análisis y al diseño del tercer módulo del proyecto: el caso de uso de una tienda en línea.

Es importante resaltar que este tercer módulo es un **caso de prueba**, por lo que solo se implementarán las acciones necesarias para que se comunique con los módulos anteriores: guardar la información bancaria tokenizada de los clientes y realizar compras donde el servicio de tokenización fingirá realizar las transacciones bancarias correspondientes.

Finalmente, se ha decidido que, para aterrizar el caso de prueba, la tienda en línea venderá libros; en otras palabras, será el portal de una librería.

8.1. Análisis

En esta sección se utilizarán, como en el capítulo correspondiente al análisis de la aplicación web, **negritas** para referirse a estados (**tiene editor, con descuento**) o tipos de un elemento (**entrega, facturación** para tipos de dirección) e **itálicas** para referirse a elementos de las interfaces.

8.1.1. Casos de uso

A continuación se detallan los casos de uso de la tienda en línea. En el diagrama 8.1 se muestra una vista general de todos junto con su relación con los actores (**visitante** o **cliente**) de la tienda.

CULIB-01 Iniciar sesión.

Caso que permite que un **visitante** se identifique ante el sistema. El **visitante** introduce su correo electrónico y contraseña y, dependiendo de dónde se encontraba (en inicio o finalizando una compra) es redirigido a la página correspondiente.

Trayectoria principal.

1. El visitante presiona el botón *Iniciar sesión*, en la pantalla principal IULIB-01 PÁGINA DE INICIO o *Finalizar compra* en IULIB-10 PÁGINA DE FINALIZAR UNA COMPRA sin haber iniciado sesión.
2. El sistema muestra la pantalla IULIB-03 FORMULARIO DE INICIO DE SESIÓN con el botón *iniciar sesión* deshabilitado.
3. El visitante introduce su correo electrónico y contraseña; [01A: EL VISITANTE CANCELA LA OPERACIÓN].
4. El sistema habilita el botón *iniciar sesión*.
5. El visitante presiona el botón *iniciar sesión*; [01A: EL VISITANTE CANCELA LA OPERACIÓN].
6. El sistema valida las credenciales del usuario; [01B: CREDENCIALES INCORRECTAS, 01C: CLIENTE NO VERIFICADO].

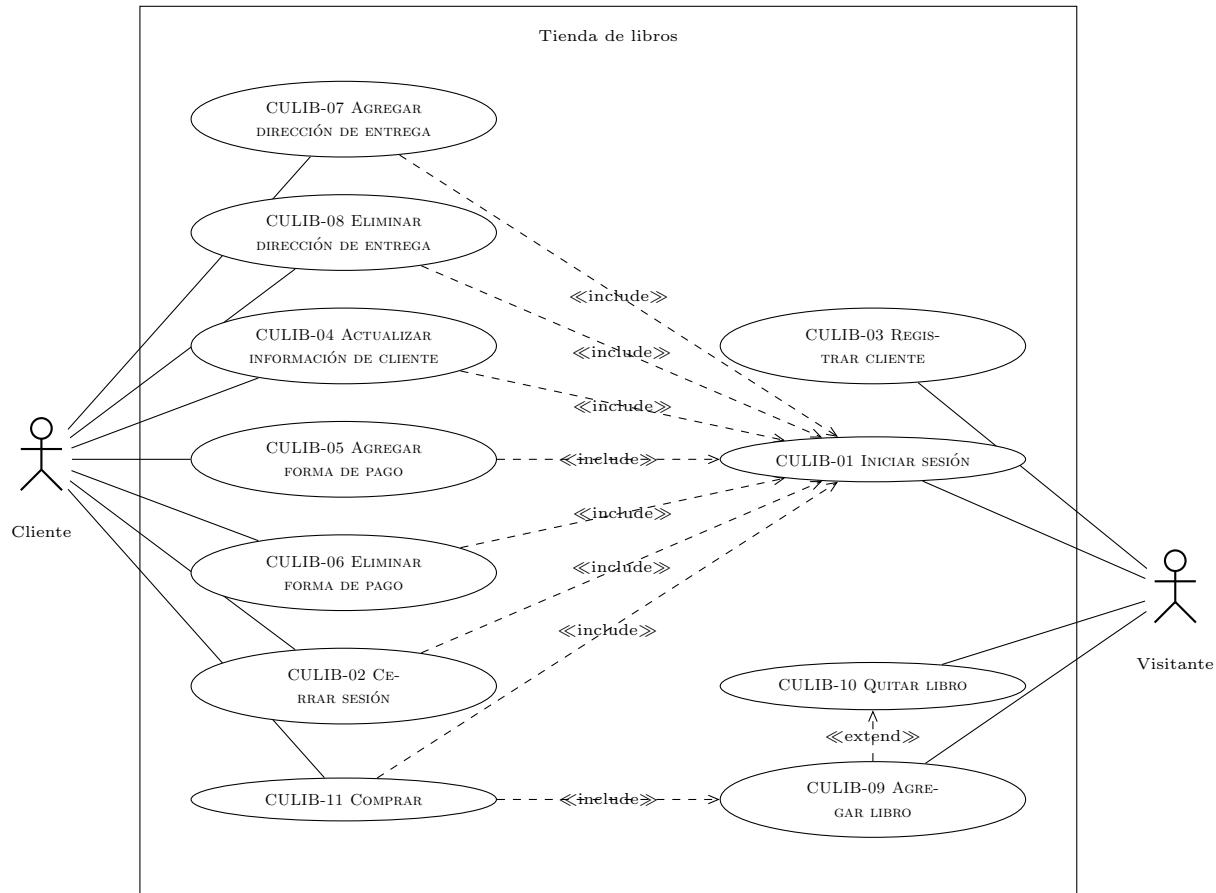


Figura 8.1: Diagrama general de casos de uso de tienda de libros en línea.

7. El sistema registra al usuario en las variables de sesión.
8. El sistema verifica que el usuario no esté realizando una compra [01D: EL CLIENTE ESTABA FINALIZANDO UNA COMPRA].
9. El sistema muestra la pantalla IULIB-01 PÁGINA DE INICIO.

Trayectoria alternativa 01A: El visitante cancela la operación.

1. El visitante presiona el botón *cancelar*.
2. El sistema muestra la interfaz de origen del paso 1 de la trayectoria principal.

Trayectoria alternativa 01B: Credenciales incorrectas.

1. El correo electrónico no se encuentra registrado en el sistema o la contraseña dada no corresponde al correo electrónico dado.
2. El sistema muestra el mensaje MSJLIB-01 CREDENCIALES INCORRECTAS.
3. Se regresa al paso 2 de la trayectoria principal.

Trayectoria alternativa 01C: Cliente no verificado.

1. El sistema muestra el mensaje MSJLIB-02 CORREO NO VERIFICADO.
2. Se regresa al paso 3 de la trayectoria principal.

Trayectoria alternativa 01D: El cliente estaba finalizando una compra.

1. El sistema obtiene los datos de todos los libros que se encuentran en el carrito del usuario.
2. El sistema arma y muestra la interfaz IULIB-10 PÁGINA DE FINALIZAR UNA COMPRA con los datos obtenidos en el paso anterior.

CULIB-02 Cerrar sesión.

Caso que permite que un **cliente** cierre su sesión en el sistema. El **cliente** presiona el botón correspondiente en la interfaz en la que se encuentra y el sistema elimina el registro correspondiente en la sesión.

Trayectoria principal.

1. El **cliente** presiona el botón de *cerrar sesión* en cualquier interfaz en la que se encuentre.
2. El sistema elimina el registro correspondiente en las variables de sesión.
3. El sistema muestra la pantalla IULIB-01 PÁGINA DE INICIO.

CULIB-03 Registrar cliente.

Caso que permite que un **visitante** se registre en el sistema, permiténdole identificarse como un **cliente** una vez terminado el proceso e iniciado sesión. Debe proveer su nombre, correo electrónico y una contraseña.

Trayectoria principal.

1. El **visitante** presiona el botón de *Registrarse* en cualquier interfaz en la que se encuentre.
2. El sistema muestra la interfaz IULIB-04 FORMULARIO DE REGISTRO con el botón *Aceptar* inhabilitado.
3. El **visitante** introduce los siguientes datos [03A: EL VISITANTE CANCELA LA OPERACIÓN]:
 - Nombre
 - Correo electrónico
 - Contraseña (y su confirmación)
4. El sistema habilita el botón de *aceptar*.
5. El **cliente** presiona el botón de *aceptar*.
6. El sistema valida que el correo electrónico sea válido al cumplir con el REQUEST FOR COMMENTS (RFC) 5322 [52], mediante la expresión regular

```
/^(([^\<>()\[\]\\.,;:\s@"]+(\.\[^<>()\[\]\\.,;:\s@"]+)*|(".+"))@((\[[0-9]{1,3}\.\[0-9]{1,3}\.\[0-9]{1,3}\.\[0-9]{1,3}\])|(([a-zA-Z\-\-0-9]+\.)+[a-zA-Z]{2,}))$/
```

[03B: CORREO ELECTRÓNICO INCORRECTO].

7. El sistema valida que el correo electrónico no esté ya registrado en el sistema; pues dos **usuarios** no pueden compartir el mismo correo; [03E: CORREO REGISTRADO PREVIAMENTE].
8. El sistema valida el formato de la contraseña de acuerdo con la expresión regular

```
/^[\x20-\x7F]{8,24}$/
```

[03C: CONTRASEÑA INCORRECTA].

9. El sistema verifica que la confirmación de contraseña ingresada coincida con la contraseña. [03D: CONFIRMACIÓN DE CONTRASEÑA INCORRECTA].
10. El sistema crea un nuevo cliente con estado **no verificado** con la información ingresada y un nuevo vínculo de verificación (asociado al cliente) con la fecha actual.
11. El sistema envía un correo al cliente con el hipervínculo de verificación.
12. El sistema muestra la ventana IULIB-05 VENTANA DE AVISO DE CORREO DE CONFIRMACIÓN.

13. El **cliente** accede a la UNIFORM RESOURCE LOCATOR (URL) enviada a su correo.
14. El sistema verifica que no haya más de 72 HRS de diferencia entre la fecha de creación del vínculo y la hora actual; [03F: VÍNCULO EXPIRADO].
15. El sistema cambia el estado del **cliente** a **verificado**.
16. El sistema muestra la pantalla IULIB-01 PÁGINA DE INICIO.

Trayectoria alternativa 03A: El visitante cancela la operación.

1. El visitante presiona el botón *cancelar*.
2. El sistema muestra la interfaz de origen del paso 1 de la trayectoria principal.

Trayectoria alternativa 03B: Correo electrónico incorrecto.

1. El sistema muestra el mensaje MSJLIB-03 CORREO ELECTRÓNICO INVÁLIDO.
2. Se regresa al paso 3 de la trayectoria principal.

Trayectoria alternativa 03C: Contraseña incorrecta.

1. El sistema muestra el mensaje MSJLIB-04 CONTRASEÑA INCORRECTA.
2. Se regresa al paso 3 de la trayectoria principal.

Trayectoria alternativa 03D: Confirmación de contraseña incorrecta.

1. El sistema muestra el mensaje MSJLIB-05 CONFIRMACIÓN DE CONTRASEÑA INCORRECTA.
2. Se regresa al paso 3 de la trayectoria principal.

Trayectoria alternativa 03E: Correo registrado previamente.

1. El sistema muestra el mensaje MSJLIB-06 CORREO REGISTRADO PREVIAMENTE.
2. Se regresa al paso 3 de la trayectoria principal.

Trayectoria alternativa 03F: Vínculo expirado.

1. El sistema muestra el mensaje .
2. El sistema elimina el registro del usuario.

CULIB-04 Actualizar información de cliente.

Caso que permite que un **cliente** actualice su información en el sistema. Puede actualizar los siguientes datos:

- Nombre
- Correo electrónico

- Contraseña

Trayectoria principal.

1. El **cliente** presiona el botón de *Actualizar información* en la interfaz IULIB-06 PÁGINA CON LOS DETALLES DE LA CUENTA DE UN USUARIO.
2. El sistema muestra la interfaz IULIB-07 FORMULARIO DE INFORMACIÓN PERSONAL con la información del cliente, y el botón *Guardar cambios* inhabilitado.
3. El **cliente** actualiza los siguientes campos [04A: EL VISITANTE CANCELA LA OPERACIÓN]:

- Nombre
- Correo electrónico
- Contraseña (y su confirmación)

4. El sistema habilita el botón de *aceptar*.
5. El **cliente** presiona el botón de *aceptar*.
6. EL sistema valida que el correo electrónico sea válido al cumplir con el RFC 5322 [52], mediante la expresión regular

```
/^(((^<>()\\[\\]\\\\.,;:\\s@"]+\\(.[^<>()\\[\\]\\\\.,;:\\s@"]+)*|(".+"))@  
((\\[[0-9]{1,3}\\.[0-9]{1,3}\\.[0-9]{1,3}\\.[0-9]{1,3}\\])|(([a-zA-Z\\-0-9]+\\.)+  
[a-zA-Z]{2,}))$/
```

[04B: CORREO ELECTRÓNICO INCORRECTO]

7. El sistema valida que el correo electrónico no esté ya registrado en el sistema; pues dos **usuarios** no pueden compartir el mismo correo; [04E: CORREO REGISTRADO PREVIAMENTE].
8. El sistema valida el formato de la contraseña de acuerdo con la expresión regular

```
/^[\x20-\x7F]{8,24}$/
```

[04C: CONTRASEÑA INCORRECTA].

9. El sistema verifica que la confirmación de contraseña ingresada coincida con la contraseña. [04D: CONFIRMACIÓN DE CONTRASEÑA INCORRECTA].
10. El sistema registra el nuevo nombre de usuario, el nuevo correo, la nueva contraseña y un nuevo vínculo de verificación (asociado al cliente) con la fecha actual, cambiando el estado a **no verificado**.
11. El sistema envía un correo al cliente con el hipervínculo de verificación.
12. El sistema muestra la ventana IULIB-05 VENTANA DE AVISO DE CORREO DE CONFIRMACIÓN.

13. El **cliente** accede a la URL enviada a su correo.
14. El sistema cambia el estado del **cliente** a **verificado**.
15. El sistema muestra la pantalla IULIB-06 PÁGINA CON LOS DETALLES DE LA CUENTA DE UN USUARIO.

Trayectoria alternativa 04A: El visitante cancela la operación.

1. El visitante presiona el botón *cancelar*.
2. El sistema muestra la interfaz de origen del paso 1 de la trayectoria principal.

Trayectoria alternativa 04B: Correo electrónico incorrecto.

1. El sistema muestra el mensaje MSJLIB-03 CORREO ELECTRÓNICO INVÁLIDO.
2. Se regresa al paso 3 de la trayectoria principal.

Trayectoria alternativa 04C: Contraseña incorrecta.

1. El sistema muestra el mensaje MSJLIB-04 CONTRASEÑA INCORRECTA.
2. Se regresa al paso 3 de la trayectoria principal.

Trayectoria alternativa 04D: Confirmación de contraseña incorrecta.

1. El sistema muestra el mensaje MSJLIB-05 CONFIRMACIÓN DE CONTRASEÑA INCORRECTA.
2. Se regresa al paso 3 de la trayectoria principal.

Trayectoria alternativa 04E: Correo registrado previamente.

1. El sistema muestra el mensaje MSJLIB-06 CORREO REGISTRADO PREVIAMENTE.
2. Se regresa al paso 3 de la trayectoria principal.

CULIB-05 Agregar forma de pago.

Caso que permite que un **cliente** agregue una forma de pago; debe proporcionar la siguiente información relacionada a la tarjeta a agregar:

- Nombre del titular de la tarjeta
- Dirección de facturación de la tarjeta
- Tipo de la tarjeta (crédito/débito)
- Emisor de la tarjeta (American Express, VISA, MasterCard, ...)
- Número de la tarjeta

- Fecha de vencimiento

Ya que este es un caso de prueba para el sistema tokenizador, el **cliente** puede escoger el método con el que se creará el token. En un ambiente de producción, es más probable que todos sus tokenizaciones sean hechas con un mismo algoritmo.

El sistema no guardará el número de tarjeta, sino que realizará una operación de tokenización con el número de tarjeta en el servicio tokenizador y guardará el token que regrese como resultado y el algoritmo que fue utilizado.

El sistema también debe mostrar un aviso que aclare qué servicio de tokenización se está utilizando y aclarar que la tienda no guardará el número de tarjeta (sino el sistema tokenizador).

Trayectoria principal.

1. El **cliente** presiona el botón de *Agregar forma de pago* en la interfaz IULIB-06 PÁGINA CON LOS DETALLES DE LA CUENTA DE UN USUARIO, en IULIB-11 PÁGINA DE SELECCIÓN DE LA FORMA DE PAGO PARA UNA COMPRA o presiona *Finalizar compra* en IULIB-10 PÁGINA DE FINALIZAR UNA COMPRA sin tener formas de pago asociadas.
2. El sistema muestra la interfaz IULIB-08 FORMULARIO DE INFORMACIÓN BANCARIA con el botón *Agregar tarjeta* inhabilitado y una lista con las direcciones que se tienen registradas (si se tienen direcciones registradas).
3. El **cliente** ingresa la siguiente información; [05A: EL **CLIENTE** SELECCIONA UNA DE LAS DIRECCIONES YA GUARDADAS, 05B: EL **CLIENTE** CANCELA LA OPERACIÓN]:
 - Nombre del titular de la tarjeta
 - Tipo de la tarjeta (crédito/débito)
 - Emisor de la tarjeta (American Express, VISA, MasterCard, ...)
 - Número de la tarjeta
 - Algoritmo tokenizador
 - Fecha de vencimiento
 - Dirección de facturación de la tarjeta
4. El sistema habilita el botón *Agregar tarjeta*.
5. El **cliente** presiona el botón *Agregar tarjeta*; [05B: EL **CLIENTE** CANCELA LA OPERACIÓN].
6. El sistema valida que el número de la tarjeta sea válido tomando en cuenta la siguiente expresión regular y asegurándose de que el dígito verificador sea correcto mediante el algoritmo de Luhn:

/^ [0-9] {12,19} \$/

[05C: LA TARJETA INGRESADA ES INVÁLIDA].

7. El sistema valida que la fecha de vencimiento no sea menor o igual a la fecha actual; [05D: LA TARJETA INGRESADA ESTÁ EXPIRADA].
8. El sistema verifica que no exista otra tarjeta relacionada al **cliente** que tenga el mismo tipo, emisor, titular y últimos cuatro dígitos; [05E: LA TARJETA INGRESADA YA HA SIDO ALMACENADA].
9. El sistema envía la petición de tokenización del número de tarjeta ingresado y el algoritmo seleccionado al servicio de tokenización; [05G: EL CÓDIGO HTTP DE RESPUESTA NO ES UN 2XX].
10. El sistema crea un nuevo registro de forma de pago con el token, la información proporcionada en el formulario y estado **activo**.
11. El sistema muestra el mensaje MSJLIB-11 MÉTODO DE PAGO AGREGADO.
12. El sistema muestra la interfaz de origen del paso 1 de la trayectoria principal.

Trayectoria alternativa 05A: El cliente selecciona una de las direcciones ya guardadas.

1. El **cliente** selecciona una de las direcciones ya guardadas.
2. El sistema completa con la información de la dirección seleccionada los campos de la dirección.
3. Se regresa al paso 3 de la trayectoria principal.

Trayectoria alternativa 05B: El cliente cancela la operación.

1. El **cliente** presiona el botón *cancelar*.
2. El sistema muestra la interfaz de origen del paso 1 de la trayectoria principal.

Trayectoria alternativa 05C: La tarjeta ingresada es inválida.

1. El sistema muestra el mensaje MSJLIB-07 TARJETA INVÁLIDA.
2. Se regresa al paso 3 de la trayectoria principal.

Trayectoria alternativa 05D: La tarjeta ingresada está expirada.

1. El sistema muestra el mensaje MSJLIB-08 TARJETA EXPIRADA.
2. Se regresa al paso 3 de la trayectoria principal.

Trayectoria alternativa 05E: La tarjeta ingresada ya ha sido almacenada.

1. El sistema verifica que el registro de la tarjeta existente sea **activo**; [05F: LA TARJETA INGRESADA YA HA SIDO ALMACENADA Y SE ENCUENTRA INACTIVA].
2. El sistema verifica que la fecha de vencimiento ingresada coincida con la que se tiene almacenada; [05H: HAY UNA TARJETA EXISTENTE Y ACTIVA CON LOS MISMOS DATOS, EXCEPTO LA FECHA DE VENCIMIENTO].
3. El sistema muestra el mensaje MSJLIB-09 TARJETA EXISTENTE.

4. El sistema muestra la interfaz de origen del paso 1 de la trayectoria principal.

Trayectoria alternativa 05F: La tarjeta ingresada ya ha sido almacenada y se encuentra inactiva.

1. El sistema actualiza el registro de la tarjeta con la fecha de vencimiento y dirección ingresada.
2. El sistema actualiza el registro del estado de la tarjeta a **activo**.
3. El sistema muestra el mensaje MSJLIB-11 MÉTODO DE PAGO AGREGADO.
4. El sistema muestra la interfaz de origen del paso 1 de la trayectoria principal.

Trayectoria alternativa 05G: El código HTTP de respuesta no es un 2XX.

1. El sistema muestra el mensaje MSJLIB-10 ERROR AL TOKENIZAR.
2. El sistema muestra la interfaz de origen del paso 1 de la trayectoria principal.

Trayectoria alternativa 05H: Hay una tarjeta existente y activa con los mismos datos, excepto la fecha de vencimiento.

1. El sistema muestra el mensaje MSJLIB-12 DOS TARJETAS CON EL MISMO NÚMERO Y DISTINTA FECHA DE VENCIMIENTO.
2. El sistema muestra la interfaz de origen del paso 1 de la trayectoria principal.

CULIB-06 Eliminar forma de pago.

Caso que permite que un **cliente** elimine una de las formas de pago que tiene guardadas. Para mantener consistencia en la base, el registro de la tarjeta no será eliminado; sino que cambiará su estado a **inactivo**.

Trayectoria principal.

1. El **cliente** presiona el botón de *Eliminar* en la interfaz .
2. El sistema muestra el mensaje emergente MSJLIB-13 ADVERTENCIA AL ELIMINAR UNA FORMA DE PAGO.
3. El **cliente** presiona *Aceptar* en el mensaje emergente; [06A: EL **CLIENTE** CANCELA LA OPERACIÓN].
4. El sistema cambia el estado de la tarjeta a **inactivo**.
5. El sistema verifica que la dirección de facturación asociada a esta forma de pago no tenga otra tarjeta y cambia su estado a **inactivo**.
6. El sistema muestra la interfaz .

Trayectoria alternativa 06A: El cliente cancela la operación.

1. El **cliente** presiona el botón *cancelar*.
2. El sistema muestra la interfaz .

CULIB-07 Agregar dirección de entrega.

Caso que permite que un **cliente** agregue una dirección de entrega.

Trayectoria principal.

1. El **cliente** presiona el botón de *Agregar dirección de entrega* en la interfaz IULIB-06 PÁGINA CON LOS DETALLES DE LA CUENTA DE UN USUARIO, *Nueva dirección* en la interfaz IULIB-12 PÁGINA DE SELECCIÓN DE LA DIRECCIÓN DE ENTREGA o presiona *Continuar* en IULIB-11 PÁGINA DE SELECCIÓN DE LA FORMA DE PAGO PARA UNA COMPRA sin tener direcciones de entrega guardadas.
2. El sistema muestra la interfaz IULIB-09 FORMULARIO DE DIRECCIÓN DE ENTREGA con el botón *Guardar dirección* inhabilitado.
3. El **cliente** ingresa la siguiente información:
 - Calle
 - Número exterior e interior
 - Colonia o localidad
 - Delegación o municipio
 - Estado
 - Código postal
4. El sistema habilita el botón *Guardar dirección*.
5. El **cliente** presiona el botón *Guardar dirección*; [07A: EL VISITANTE CANCELA LA OPERACIÓN].
6. El sistema valida que esa dirección no esté asociada ya al cliente; [07B: DIRECCIÓN YA EXISTENTE].
7. El sistema crea un nuevo registro con la dirección ingresada, estado **activo** y la asocia al cliente.
8. El sistema muestra el mensaje MSJLIB-14 NUEVA DIRECCIÓN DE ENTREGA GUARDADA
9. El sistema muestra la interfaz del paso 1 de la trayectoria principal.

Trayectoria alternativa 07A: El visitante cancela la operación.

1. El visitante presiona el botón *cancelar*.
2. El sistema muestra la interfaz de origen del paso 1 de la trayectoria principal.

Trayectoria alternativa 07B: Dirección ya existente.

1. El sistema verifica que la dirección tenga estado **activo**; [07C: DIRECCIÓN EXISTENTE PERO INACTIVA].
2. El sistema muestra el mensaje MSJLIB-15 DIRECCIÓN DE ENTREGA EXISTENTE.
3. Se regresa al paso 3 de la trayectoria principal.

Trayectoria alternativa 07C: Dirección existente pero inactiva.

1. El sistema cambia el estado de la dirección a **activo**.
2. Se regresa al paso 8 de la trayectoria principal.

CULIB-08 Eliminar dirección de entrega.

Caso que permite que un **cliente** elimine una de las direcciones de entrega que tiene guardadas. Para mantener consistencia en la base, el registro de la dirección no será eliminado; sino que cambiará su estado a **inactivo**.

Trayectoria principal.

1. El **cliente** presiona el botón de *Eliminar dirección* en la interfaz .
2. El sistema muestra el mensaje emergente MSJLIB-16 ADVERTENCIA AL ELIMINAR UNA DIRECCIÓN DE ENTREGA.
3. El **cliente** presiona *Aceptar* en el mensaje emergente; [08A: EL **CLIENTE** CANCELA LA OPERACIÓN].
4. El sistema cambia el estado de la tarjeta a **inactivo**.
5. El sistema muestra la interfaz .

Trayectoria alternativa 08A: El cliente cancela la operación.

1. El **cliente** presiona el botón *cancelar*.
2. El sistema muestra la interfaz .

CULIB-09 Agregar libro.

Caso que permite que un **cliente** o un **visitante** agregue libros al carrito de compra.

Trayectoria principal.

1. El usuario presiona el botón *Agregar al carrito* en cualquiera de estas interfaces:
 - En IULIB-01 PÁGINA DE INICIO, en el botón correspondiente al libro deseado.
 - En IULIB-02 PÁGINA DE DETALLES DE UN LIBRO.
2. El sistema actualiza la interfaz para reflejar que el libro seleccionado se encuentra en el carrito.

3. El sistema guarda el libro en los datos de sesión.
4. Si la pantalla de origen del paso 1 es IULIB-02 PÁGINA DE DETALLES DE UN LIBRO el sistema muestra un entrada numérica para introducir la cantidad de ejemplares a comprar (con un valor por defecto de uno); [09A: EL USUARIO MODIFICA LA CANTIDAD POR DEFECTO].

Trayectoria alternativa 09A: El usuario modifica la cantidad por defecto.

1. El usuario introduce la cantidad de ejemplares del libro deseado.
2. El sistema verifica que las existencias del libro seleccionado sean suficientes para la cantidad de ejemplares introducida por el usuario; [09B: EJEMPLARES INSUFICIENTES].
3. El sistema guarda el dato introducido en los datos de sesión.

Trayectoria alternativa 09B: Ejemplares insuficientes.

1. El sistema muestra el mensaje MSJLIB-18 EJEMPLARES INSUFICIENTES.
2. Se regresa al paso 1 de la trayectoria anterior.

CULIB-10 Quitar libro.

Caso que permite que un **cliente** o un **visitante** quiten libros del carrito de compra.

Trayectoria principal.

1. El usuario presiona el botón *Quitar del carrito* en cualquiera de estas interfaces:
 - En IULIB-01 PÁGINA DE INICIO, en el botón correspondiente al libro que se desea quitar.
 - En IULIB-02 PÁGINA DE DETALLES DE UN LIBRO.
 - En IULIB-10 PÁGINA DE FINALIZAR UNA COMPRA, en el botón correspondiente al libro que se desea quitar.
2. El sistema actualiza la interfaz para reflejar que el libro seleccionado ya no se encuentra en el carrito.
3. El sistema elimina el registro correspondiente en los datos de sesión.

CULIB-11 Comprar.

Caso que permite que un **cliente** realice la compra de los libros que hay en el carrito.

Como caso de prueba del sistema tokenizador, al momento de registrar la compra, se hace una operación de detokenización para simular la posible interacción con el entorno bancario. Una implementación real se podría hacer de dos maneras:

- El sistema tokenizador funciona como intermediario entre la tienda y el entorno bancario. En este caso, en lugar de la operación de detokenización, se haría una petición de transacción.
- Es la tienda la que se ocupa de interactuar con el banco. En este caso, después de la operación de detokenización, la tienda tendría que implementar la transacción bancaria.

Trayectoria principal.

1. El cliente presiona el botón *Finalizar compra* en la interfaz IULIB-10 PÁGINA DE FINALIZAR UNA COMPRA.
2. El sistema obtiene las formas de pago del cliente en sesión.
3. El sistema muestra la interfaz IULIB-11 PÁGINA DE SELECCIÓN DE LA FORMA DE PAGO PARA UNA COMPRA.
4. El cliente selecciona la forma de pago deseada; [11A: EL CLIENTE CANCELA LA OPERACIÓN].
5. El cliente presiona el botón *Continuar*; [11A: EL CLIENTE CANCELA LA OPERACIÓN].
6. El sistema muestra la interfaz IULIB-12 PÁGINA DE SELECCIÓN DE LA DIRECCIÓN DE ENTREGA.
7. El cliente selecciona la dirección de entrega deseada; [11A: EL CLIENTE CANCELA LA OPERACIÓN].
8. El cliente presiona el botón *Continuar*; [11A: EL CLIENTE CANCELA LA OPERACIÓN].
9. El sistema muestra la interfaz IULIB-13 PÁGINA DE RESUMEN DE COMPRA.
10. El cliente presiona el botón *Aceptar*; [11A: EL CLIENTE CANCELA LA OPERACIÓN].
11. El sistema envía la petición de detokenización del token del cliente en sesión¹.
12. El sistema registra la compra y resta de las existencias los libros comprados.
13. El sistema muestra el mensaje MSJLIB-17 COMPRA REGISTRADA.

Trayectoria alternativa 11A: El cliente cancela la operación.

1. El cliente presiona el botón *cancelar*.
2. El sistema muestra la interfaz IULIB-10 PÁGINA DE FINALIZAR UNA COMPRA.

8.1.2. Interfaces de usuario

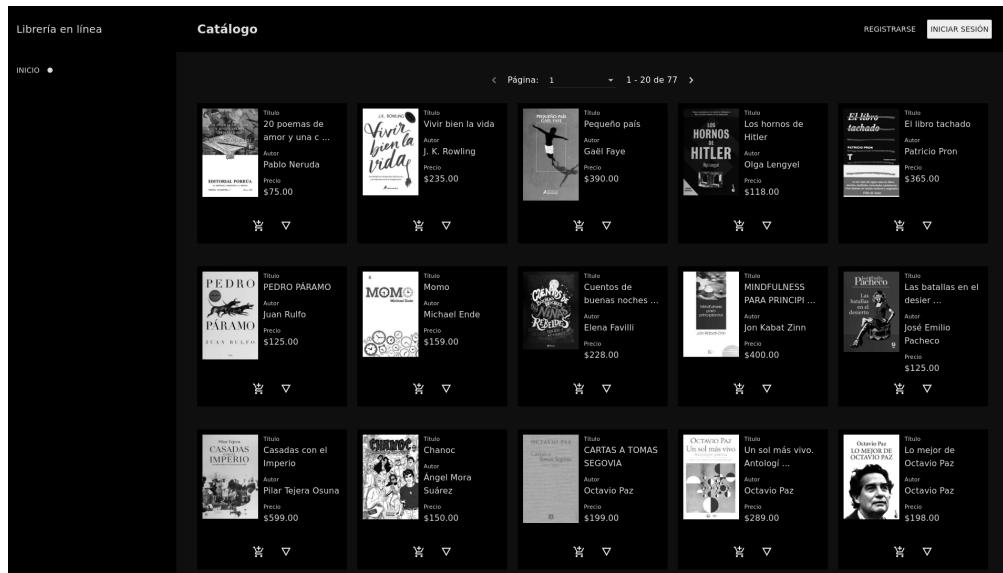
8.1.3. Catálogo de mensajes

MSJLIB-01 Credenciales incorrectas.

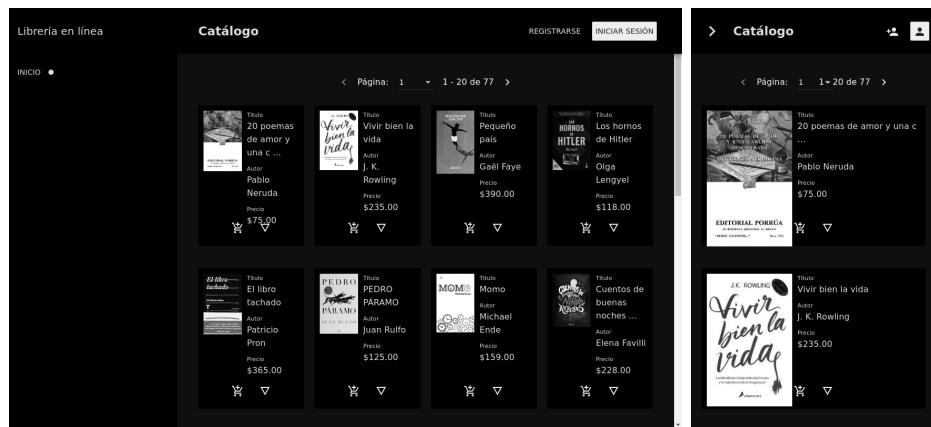
Nombre de usuario y/o contraseña incorrecta.

MSJLIB-02 Correo no verificado.

¹Después de este paso es en donde una tienda real tendría que hacer la transacción bancaria.

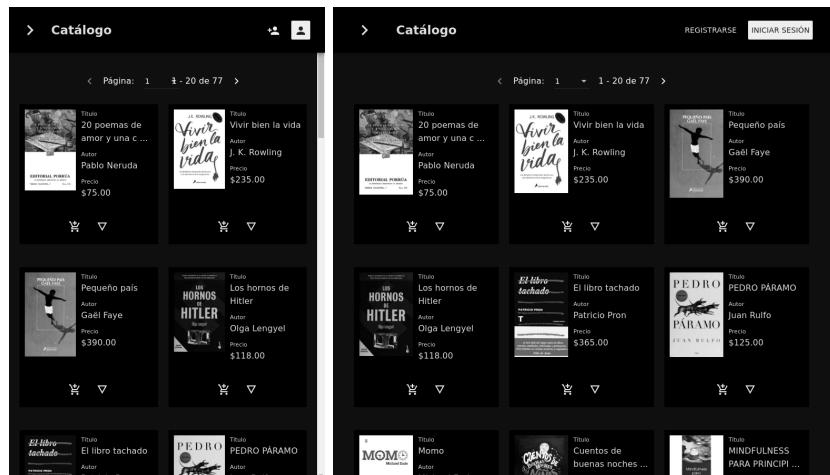


(a) 1920 x 1080



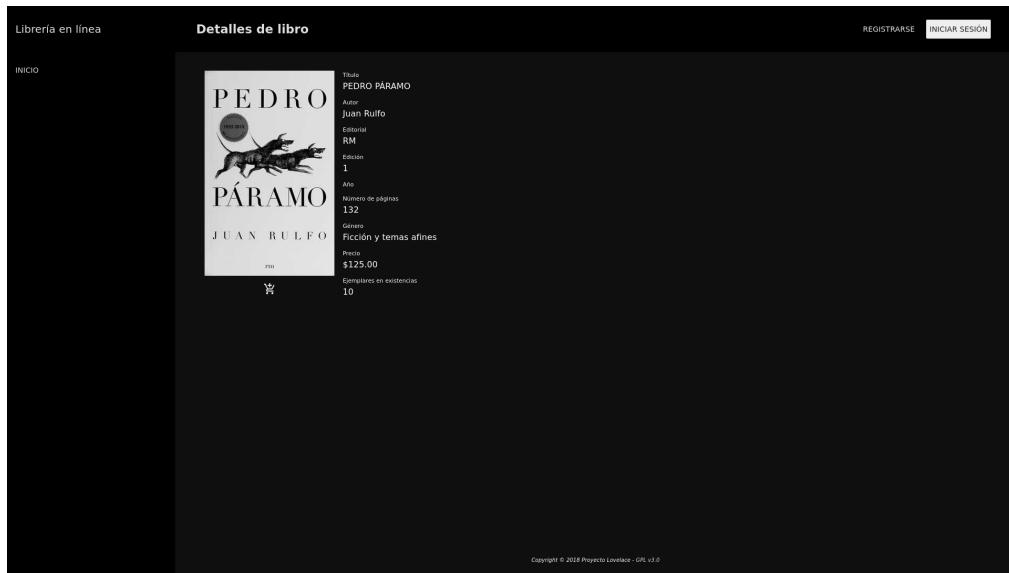
(b) 1280 x 800

(c) 480 x 800



(d) 600 x 960

(e) 960 x 900



(a) 1920 x 1080

Detalles de libro

PEDRO PÁRAMO

JUAN RULFO

Título: PEDRO PÁRAMO
Autor: Juan Rulfo
Editorial: RM
Edición: 1
Año:
Número de páginas: 132
Género: Ficción y temas afines
Precio: \$125.00
Ejemplares en existencias: 10

Detalles de libro

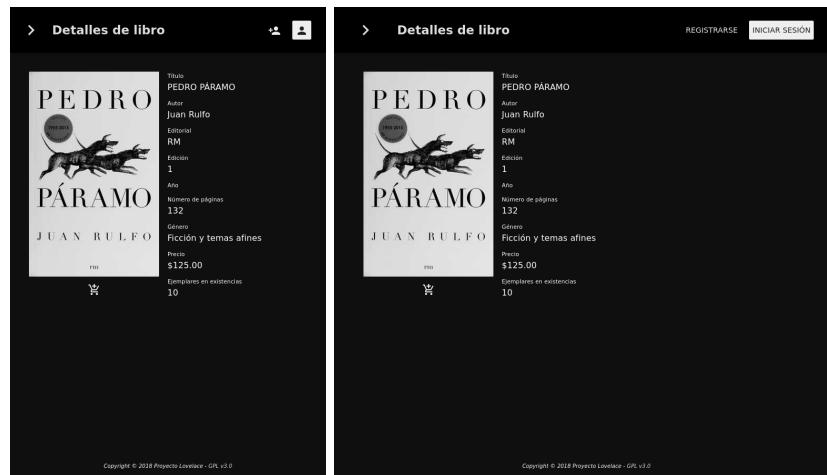
PEDRO PÁRAMO

JUAN RULFO

Título: PEDRO PÁRAMO
Autor: Juan Rulfo
Editorial: RM
Edición: 1
Año:
Número de páginas: 132
Género: Ficción y temas afines
Precio: \$125.00
Ejemplares en existencias: 10

(b) 1280 x 800

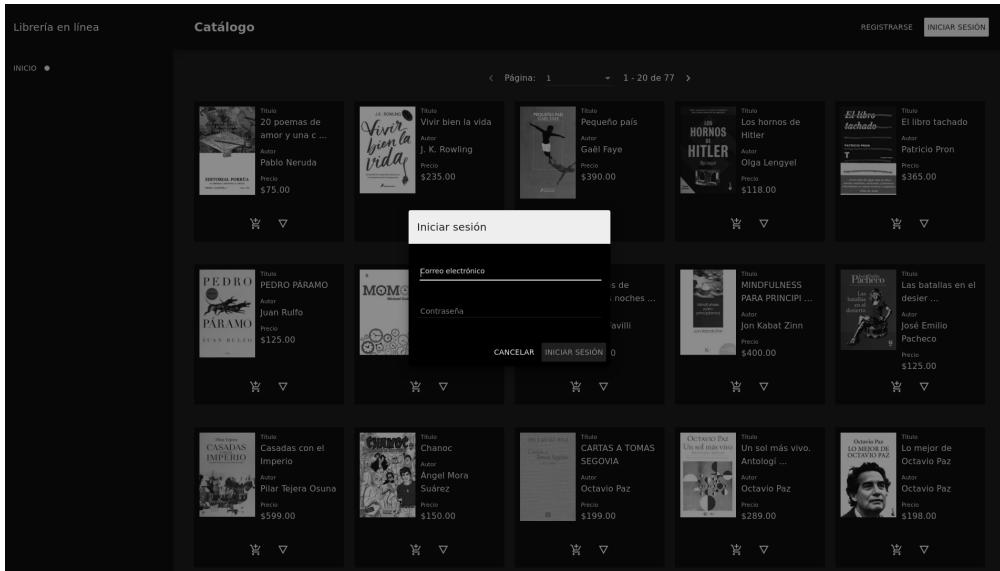
(c) 480 x 800



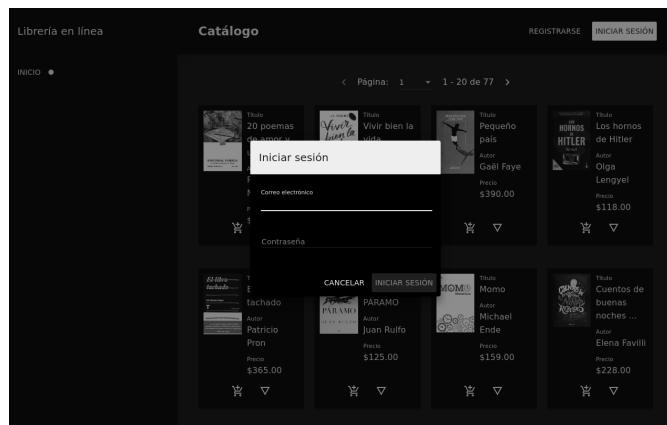
(d) 600 x 960

(e) 960 x 900

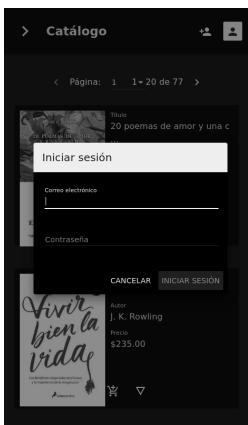
IULIB-02 Página de detalles de un libro



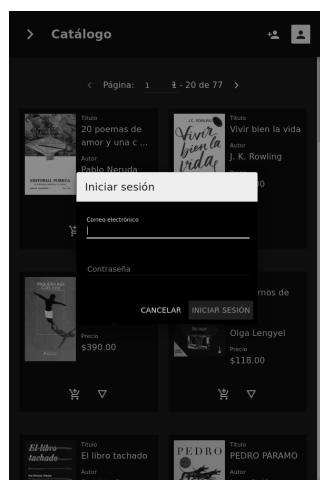
(a) 1920 x 1080



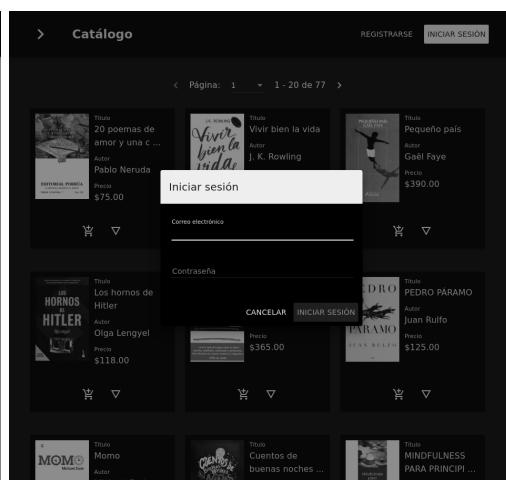
(b) 1280 x 800



(c) 480 x 800

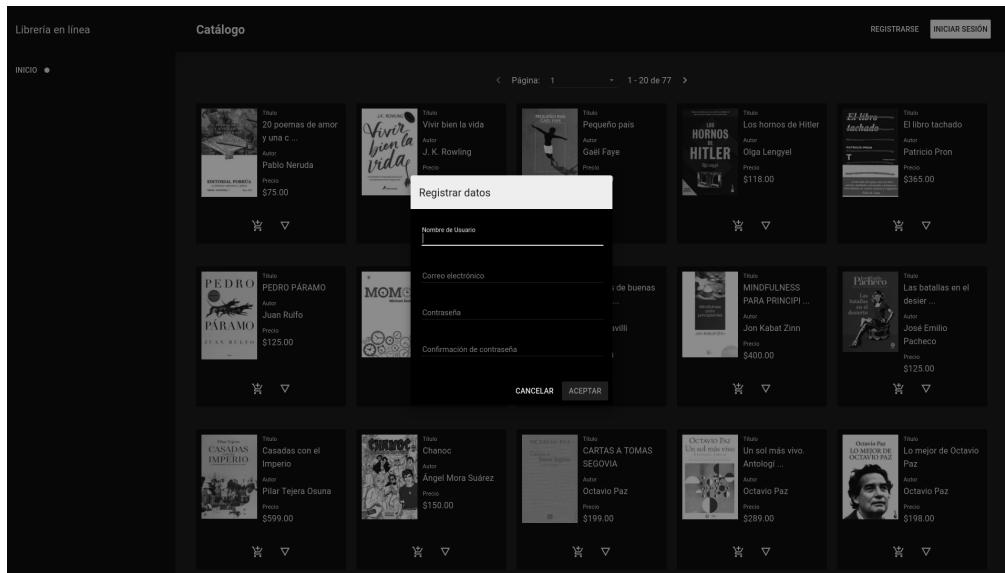


(d) 600 x 960

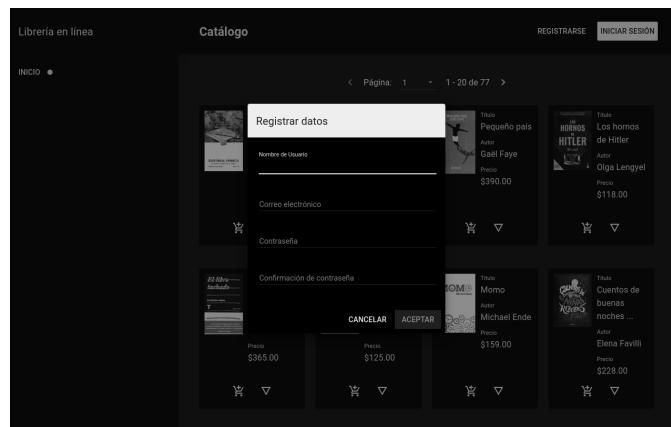


(e) 960 x 900

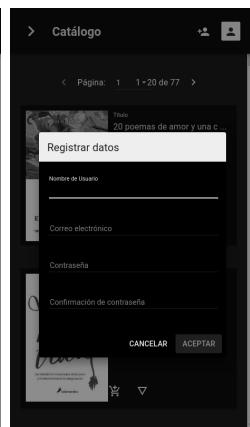
IULIB-03 Formulario de inicio de sesión



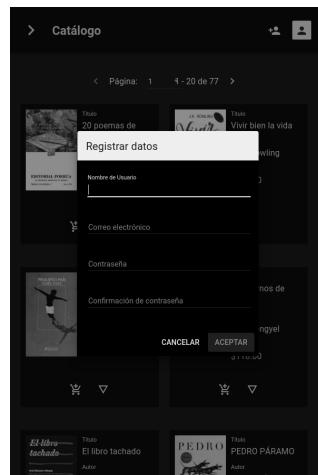
(a) 1920 x 1080



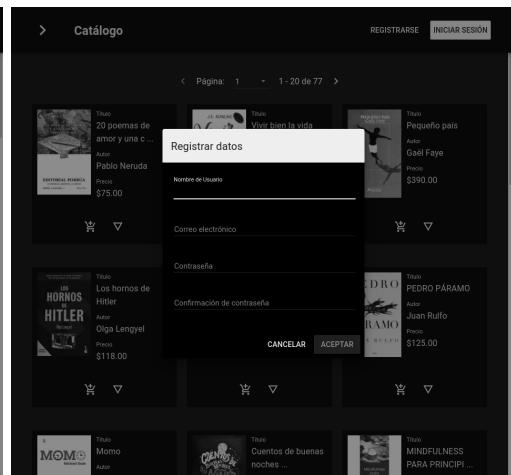
(b) 1280 x 800



(c) 480 x 800



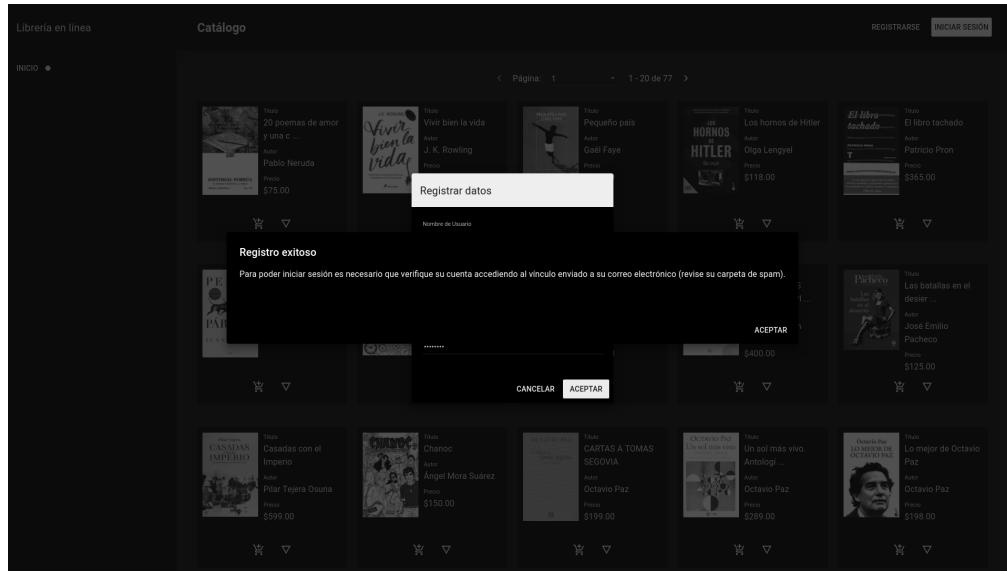
(d) 600 x 960



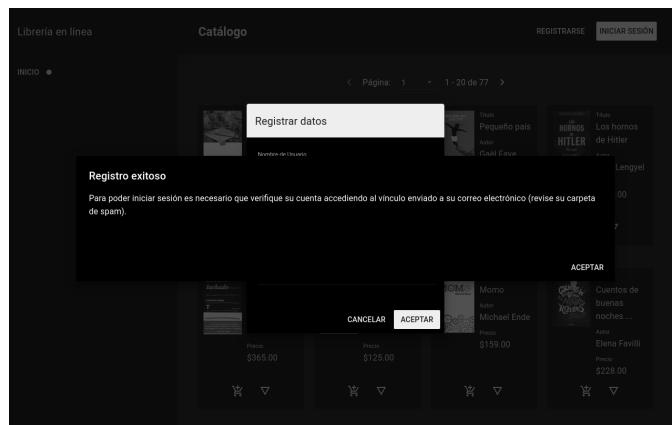
(e) 960 x 900

IULIB-04 Formulario de registro

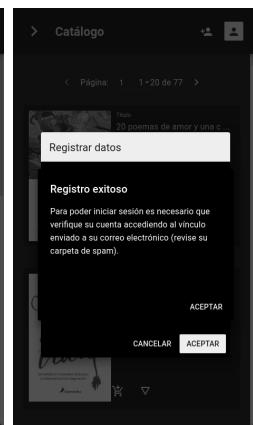
Generación de *tokens* para proteger los datos de tarjetas bancarias



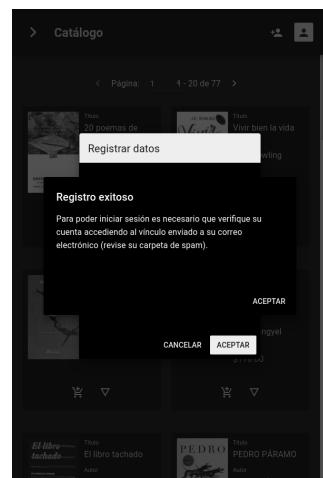
(a) 1920 x 1080



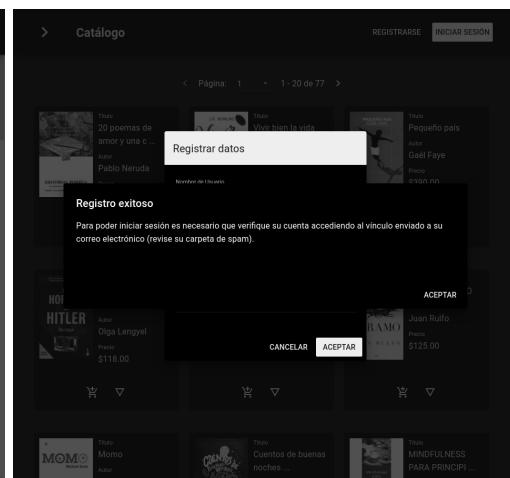
(b) 1280 x 800



(c) 480 x 800



(d) 600 x 960



(e) 960 x 900

IULIB-05 Ventana de aviso de correo de confirmación

Librería en línea **Administración de cuenta**

INICIO **CUENTA** ●

Información personal

Nombre de usuario: Cliente Correo electrónico: cliente@prueba.com

EDITAR INFORMACIÓN

Métodos de pago

Titular: Ricardo Quezada Figueroa	Calle: Lázaro Cárdenas	Titular: Nombre de Prueba	Calle: Reforma
Términación: 8765	Número interior: 8	Términación: 5698	Número interior: 34
Fecha de expiración: 10/10/2020	Número exterior: 4	Fecha de expiración: 10/10/2020	Número exterior: 98
Emisor: VISA	Colonia: Regeneración Nacional	Emisor: MasterCard	Colonia: Centro
Tipo de tarjeta: Débito	Municipio: Tapachula	Tipo de tarjeta: Débito	Municipio: Ixtapan de la Sal
Método de tokenización: FFX	Estado: Chiapas	Método de tokenización: AHR	Estado: Estado de México
Código postal: 58697		Código postal: 99835	

QUITAR METODO **QUITAR METODO**

AGREGAR METODO DE PAGO

Direcciones de entrega

Calle: Independencia	Calle: Corriente a Tetipac
Número interior:	Número interior:

LIBRERIA EN LÍNEA **ADMINISTRACIÓN DE CUENTA** **CLIENTE** **CERRAR SESIÓN**

(a) 1920 x 1080

Librería en línea **Administración de cuenta**

INICIO **CUENTA** ●

Información personal

Nombre de usuario: Cliente Correo electrónico: cliente@prueba.com

EDITAR INFORMACIÓN

Métodos de pago

Titular: Ricardo Quezada Figueroa	Calle: Lázaro Cárdenas	Titular: Nombre de Prueba	Calle: Reforma
Términación: 8765	Número interior: 8	Términación: 5698	Número interior: 34
Fecha de expiración: 10/10/2020	Número exterior: 4	Fecha de expiración: 10/10/2020	Número exterior: 98
Emisor: VISA	Colonia: Centro	Emisor: MasterCard	Colonia: Centro
Tipo de tarjeta: Débito	Municipio: Tapachula	Tipo de tarjeta: Débito	Municipio: Ixtapan de la Sal
Método de tokenización: FFX	Estado: Chiapas	Método de tokenización: AHR	Estado: Estado de México
Código postal: 58697		Código postal: 99835	

QUITAR METODO **QUITAR METODO**

AGREGAR METODO DE PAGO

Direcciones de entrega

Calle: Independencia	Calle: Corriente a Tetipac
Número interior:	Número interior:

LIBRERIA EN LÍNEA **ADMINISTRACIÓN DE CUENTA** **CLIENTE** **CERRAR SESIÓN**

(b) 1280 x 800

Librería en línea **Administración de cuenta**

INICIO **CUENTA** ●

Información personal

Nombre de usuario: Cliente Correo electrónico: cliente@prueba.com

EDITAR INFORMACIÓN

Métodos de pago

Titular: Ricardo Quezada Figueroa	Calle: Lázaro Cárdenas	Titular: Nombre de Prueba	Calle: Reforma
Términación: 8765	Número interior: 8	Términación: 5698	Número interior: 34
Fecha de expiración: 10/10/2020	Número exterior: 4	Fecha de expiración: 10/10/2020	Número exterior: 98
Emisor: VISA	Colonia: Centro	Emisor: MasterCard	Colonia: Centro
Tipo de tarjeta: Débito	Municipio: Tapachula	Tipo de tarjeta: Débito	Municipio: Ixtapan de la Sal
Método de tokenización: FFX	Estado: Chiapas	Método de tokenización: AHR	Estado: Estado de México
Código postal: 58697		Código postal: 99835	

QUITAR METODO

AGREGAR METODO DE PAGO

Direcciones de entrega

Calle: Independencia	Calle: Corriente a Tetipac
Número interior:	Número interior:

LIBRERIA EN LÍNEA **ADMINISTRACIÓN DE CUENTA** **CLIENTE** **CERRAR SESIÓN**

(c) 480 x 800

Administración de cuenta

INICIO **CUENTA** ●

Información personal

Nombre de usuario: Cliente Correo electrónico: cliente@prueba.com

EDITAR INFORMACIÓN

Métodos de pago

Titular: Ricardo Quezada Figueroa	Calle: Lázaro Cárdenas	Titular: Nombre de Prueba	Calle: Reforma
Términación: 8765	Número interior: 8	Términación: 5698	Número interior: 34
Fecha de expiración: 10/10/2020	Número exterior: 4	Fecha de expiración: 10/10/2020	Número exterior: 98
Emisor: VISA	Colonia: Centro	Emisor: MasterCard	Colonia: Centro
Tipo de tarjeta: Débito	Municipio: Tapachula	Tipo de tarjeta: Débito	Municipio: Ixtapan de la Sal
Método de tokenización: FFX	Estado: Chiapas	Método de tokenización: AHR	Estado: Estado de México
Código postal: 58697		Código postal: 99835	

QUITAR METODO **QUITAR METODO**

AGREGAR METODO DE PAGO

Direcciones de entrega

Calle: Independencia	Calle: Corriente a Tetipac
Número interior:	Número interior:

LIBRERIA EN LÍNEA **ADMINISTRACIÓN DE CUENTA** **CLIENTE** **CERRAR SESIÓN**

(d) 600 x 960

(e) 960 x 900

IULIB-06 Página con los detalles de la cuenta de un usuario

Generación de tokens para proteger los datos de tarjetas bancarias

Librería en línea Administación de cuenta Cliente CERRAR SESIÓN

INICIO

CUENTA ●

Información personal

Nombre de usuario: Cliente Correo electrónico: cliente@prueba.com

EDITAR INFORMACIÓN

Métodos de pago

Actualizar datos

Titular: Daniel Ayala	Calle: Dolo A	Nombre de Usuario: Clienté
Teléfono: 4563	Número exterior: 12	Contraseña: _____
Fecha de expiración: 01/09/2020	Número interior: 12	Confirmación de contraseña: _____
Emisor: MasterCard	Domicilio: Manzano	
Tipo de tarjeta: Débito	Municipio: El chiquito	
Método de tokenización: FFX	Estado: Campeche	
	Código postal: 04377	

QUITAR MÉTODO

AGREGAR MÉTODO DE PAGO

Direcciones de entrega

Calle: Independencia Otra: Carrerita a Tetipac

(a) 1920 x 1080

Librería en línea Administación de cuenta Cliente CERRAR SESIÓN

INICIO

CUENTA ●

Información personal

Nombre de usuario: Cliente Correo electrónico: cliente@prueba.com

EDITAR INFOR

Métodos di

Actualizar datos

Titular: Daniel Ayala	Contraseña: _____
Teléfono: 4563	Confirmación de contraseña: _____
Fecha de expiración: 01/09/2020	
Emisor: MasterCard	
Tipo de tarjeta: Débito	
Método de tokenización: FFX	

CANCELAR **ACEPTAR**

QUITAR MÉTODO

(b) 1280 x 800

> Administación de cuenta Cliente

Información personal

Nombre de usuario: Cliente Correo electrónico: cliente@prueba.com

EDITAR

Mét

Actualizar datos

Titular: Dar	Contraseña: _____
Tel: 456	Confirmación de contraseña: _____
Ema: Mbi	
Tipo de tarjeta: Débito	
Método de tokenización: FFX	

CANCELAR **ACEPTAR**

QUITAR MÉTODO

(c) 480 x 800

> Administación de cuenta Cliente

Información personal

Nombre de usuario: Cliente Correo electrónico: cliente@prueba.com

EDITAR INFORMACIÓN

Métodos de pago

Actualizar datos

Titular: Daniel Ayala	Calle: Dolo A	Nombre de Usuario: Clienté
Teléfono: 4563	Número exterior: 12	Contraseña: _____
Fecha de expiración: 01/09/2020	Número interior: 12	Confirmación de contraseña: _____
Emisor: MasterCard	Domicilio: Manzano	
Tipo de tarjeta: Débito	Municipio: El chiquito	
Método de tokenización: FFX	Estado: Campeche	
	Código postal: 04377	

CANCELAR **ACEPTAR**

QUITAR MÉTODO

AGREGAR MÉTODO DE PAGO

(d) 600 x 960

(e) 960 x 900

IULIB-07 Formulario de información personal

(a) 1920 x 1080

(b) 1280 x 800

(c) 480 x 800

(d) 600 x 960

(e) 960 x 900

IULIB-08 Formulario de información bancaria

(a) 1920 x 1080

(b) 1280 x 800

(c) 480 x 800

(d) 600 x 960

(e) 960 x 900

IULIB-09 Formulario de dirección de entrega

Capítulo 8. Análisis y diseño de tienda en línea

Librería en línea

Carrito de compra

INICIO CERRAR SESIÓN

CARRITO \$818.00 •

CUENTA

Cantidad	Libro	P. Unitario	Total	
1	20 poemas de amor y una canción desesperada · Antología Nerudiana, Pablo Neruda	\$75.00	\$75.00	X
1	Vivir bien la vida, J. K. Rowling	\$235.00	\$235.00	X
1	Pequeño país, Gaél Faye	\$390.00	\$390.00	X
1	Los hornos de Hitler, Olga Lengyel	\$118.00	\$118.00	X
Total			\$818.00	

FINALIZAR COMPA

Copyright © 2018 Proyecto Livewire - GPL v3.0

(a) 1920 x 1080

Librería en línea

Carrito de compra

Cliente CERRAR SESIÓN

CARRITO \$818.00 •

CUENTA

Cantidad	Libro	P. Unitario	Total	
1	20 poemas de amor y una canción desesperada · Antología Nerudiana, Pablo Neruda	\$75.00	\$75.00	X
1	Vivir bien la vida, J. K. Rowling	\$235.00	\$235.00	X
1	Pequeño país, Gaél Faye	\$390.00	\$390.00	X
1	Los hornos de Hitler, Olga Lengyel	\$118.00	\$118.00	X
Total			\$818.00	

FINALIZAR COMPA

Copyright © 2018 Proyecto Livewire - GPL v3.0

(b) 1280 x 800

> **Carrito de compra** Cliente

Cantidad	Libro	P. Unitario	Total
1	20 poemas de amor y una canción desesperada · Antología Nerudiana, Pablo Neruda	\$75.00	\$75.00
1	Vivir bien la vida, J. K. Rowling	\$235.00	\$235.00
1	Pequeño país, Gaél Faye	\$390.00	\$390.00
1	Los hornos de Hitler, Olga Lengyel	\$118.00	\$118.00
Total			\$818.00

FINALIZAR COMPA

Copyright © 2018 Proyecto Livewire - GPL v3.0

(c) 480 x 800

> **Carrito de compra** Cliente

Cantidad	Libro	P. Unitario	Total	
1	20 poemas de amor y una canción desesperada · Antología Nerudiana, Pablo Neruda	\$75.00	\$75.00	X
1	Vivir bien la vida, J. K. Rowling	\$235.00	\$235.00	X
1	Pequeño país, Gaél Faye	\$390.00	\$390.00	X
1	Los hornos de Hitler, Olga Lengyel	\$118.00	\$118.00	X
Total			\$818.00	

FINALIZAR COMPA

Copyright © 2018 Proyecto Livewire - GPL v3.0

(d) 600 x 960

> **Carrito de compra** Cliente CERRAR SESIÓN

Cantidad	Libro	P. Unitario	Total	
1	20 poemas de amor y una canción desesperada · Antología Nerudiana, Pablo Neruda	\$75.00	\$75.00	X
1	Vivir bien la vida, J. K. Rowling	\$235.00	\$235.00	X
1	Pequeño país, Gaél Faye	\$390.00	\$390.00	X
1	Los hornos de Hitler, Olga Lengyel	\$118.00	\$118.00	X
Total			\$818.00	

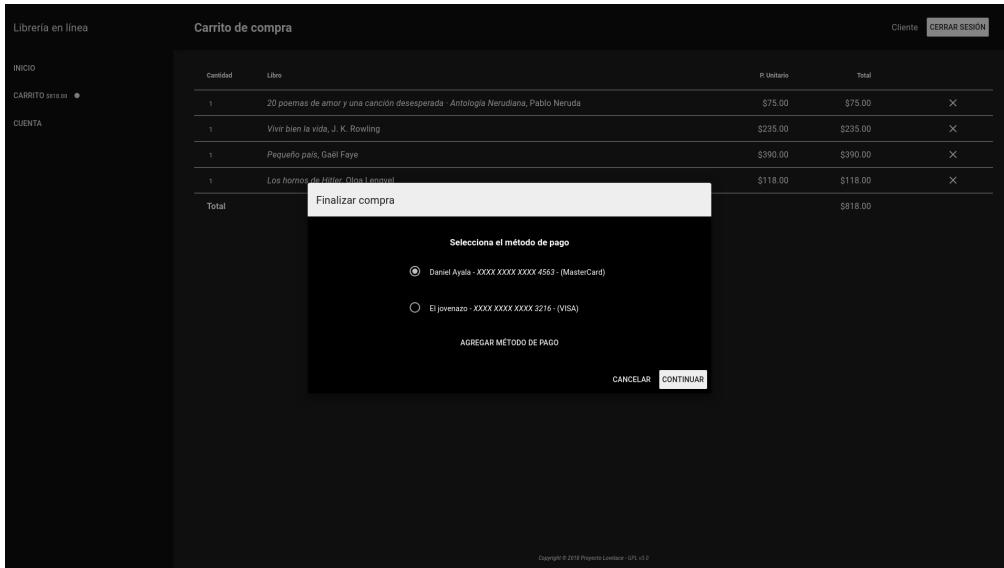
FINALIZAR COMPA

Copyright © 2018 Proyecto Livewire - GPL v3.0

(e) 960 x 900

IULIB-10 Página de finalizar una compra

Generación de tokens para proteger los datos de tarjetas bancarias



(a) 1920 x 1080

(b) 1280 x 800

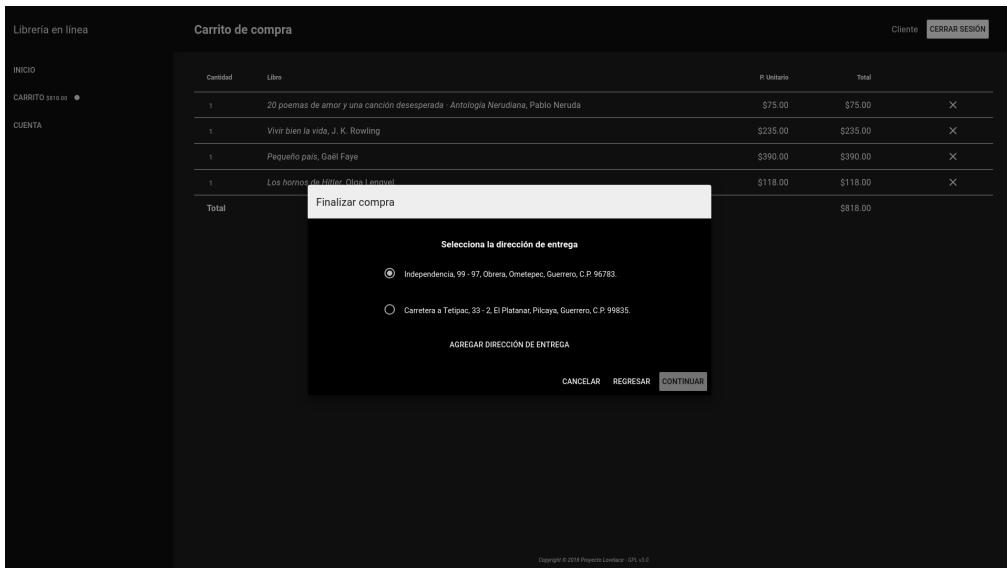
(c) 480 x 800

(d) 600 x 960

(e) 960 x 900

IULIB-11 Página de selección de la forma de pago para una compra

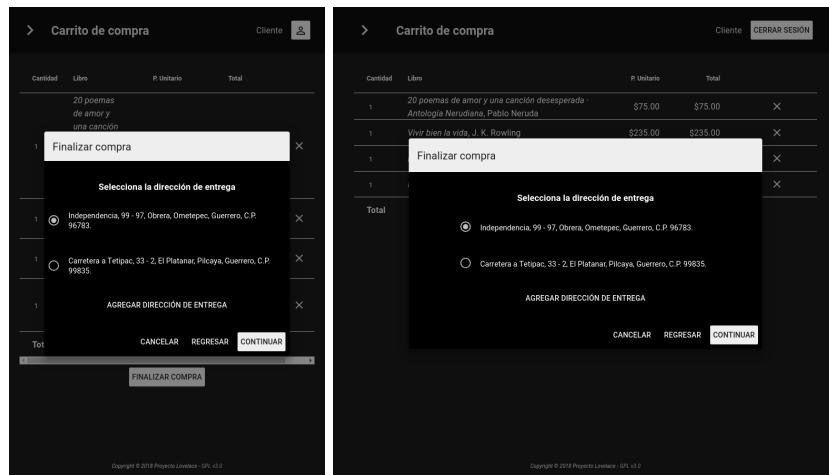
Capítulo 8. Análisis y diseño de tienda en línea



(a) 1920 x 1080

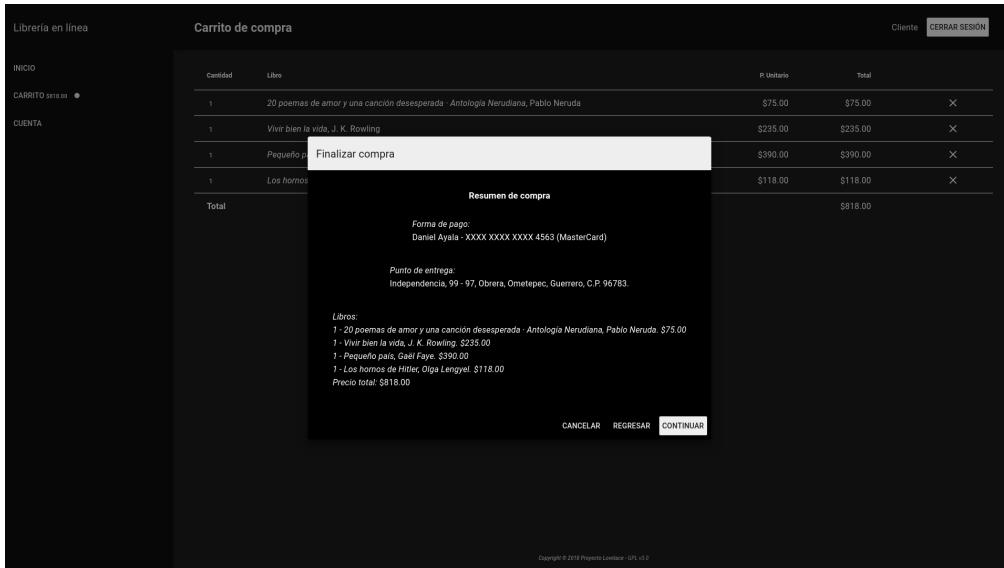
(b) 1280 x 800

(c) 480 x 800



IULIB-12 Página de selección de la dirección de entrega

Generación de tokens para proteger los datos de tarjetas bancarias



(a) 1920 x 1080

(b) 1280 x 800

(c) 480 x 800

(d) 600 x 960

(e) 960 x 900

IULIB-13 Página de resumen de compra

Este correo no está verificado; verifíquelo haciendo clic en el enlace que le fue mandado (revise su carpeta de SPAM).

MSJLIB-03 Correo electrónico inválido.

Correo electrónico inválido (RFC 5322).

MSJLIB-04 Contraseña incorrecta.

Contraseña inválida: debe contener entre 8 y 24 caracteres ASCII imprimibles.

MSJLIB-05 Confirmación de contraseña incorrecta.

La confirmación de contraseña no coincide con la contraseña ingresada.

MSJLIB-06 Correo registrado previamente.

El correo dado ya se encuentra registrado.

MSJLIB-07 Tarjeta inválida.

La tarjeta ingresada es inválida.

MSJLIB-08 Tarjeta expirada.

La tarjeta ingresada ha expirado; verifique la fecha ingresada o intente con otra.

MSJLIB-09 Tarjeta existente.

La tarjeta ingresada ya ha sido registrada.

MSJLIB-10 Error al tokenizar.

Ocurrió un error; inténtelo de nuevo más tarde.

MSJLIB-11 Método de pago agregado.

El método de pago fue agregado correctamente.

MSJLIB-12 Dos tarjetas con el mismo número y distinta fecha de vencimiento.

Usted tiene otra tarjeta con los mismos datos y una fecha de vencimiento distinta; no puede tener dos números iguales y distintas fechas de vencimiento, así que, si desea agregar este método de pago, elimine la otra forma de pago antes de intentarlo de nuevo.

MSJLIB-13 Advertencia al eliminar una forma de pago.

¿Está seguro de que quiere eliminar esta forma de pago?

MSJLIB-14 Nueva dirección de entrega guardada.

Dirección de entrega guardada exitosamente

MSJLIB-15 Dirección de entrega existente.

Esta dirección de entrega ya existe.

MSJLIB-16 Advertencia al eliminar una dirección de entrega.

¿Está seguro de que quiere eliminar esta dirección de entrega?

MSJLIB-17 Compra registrada.

Su compra ha sido registrada con el número de tarjeta: XXXX XXXX XXXX XXXX.

MSJLIB-18 Ejemplares insuficientes.

Lo sentimos, no tenemos suficientes ejemplares en existencias para satisfacer su compra.

8.2. Diseño

En esta sección se incluyen los aspectos del diseño del caso de prueba librería en línea.

8.2.1. Diseño de la base de datos

En 8.15 se puede observar el diagrama entidad relación de la tienda en línea; mientras que en la figura 8.16 se encuentra el diagrama relacional de la base de datos.

8.2.2. Vista estática

En la figura 8.17 se muestra el diagrama de clases.

8.2.3. Vista dinámica

Comenzando con los diagramas de secuencia para algunos de las acciones, en la figura 8.18 se muestra el correspondiente para el proceso de registrar a un cliente en la tienda en línea (para más detalles, véase CULIB-03 REGISTRAR CLIENTE); en la figura 8.19, el diagrama para agregar un libro al carrito; finalmente, en 8.20, se observa la secuencia para realizar una compra.

En la figura 8.21 se encuentra el diagrama de estados para un cliente, en 8.22, para una tarjeta; y en 8.23, para una dirección.

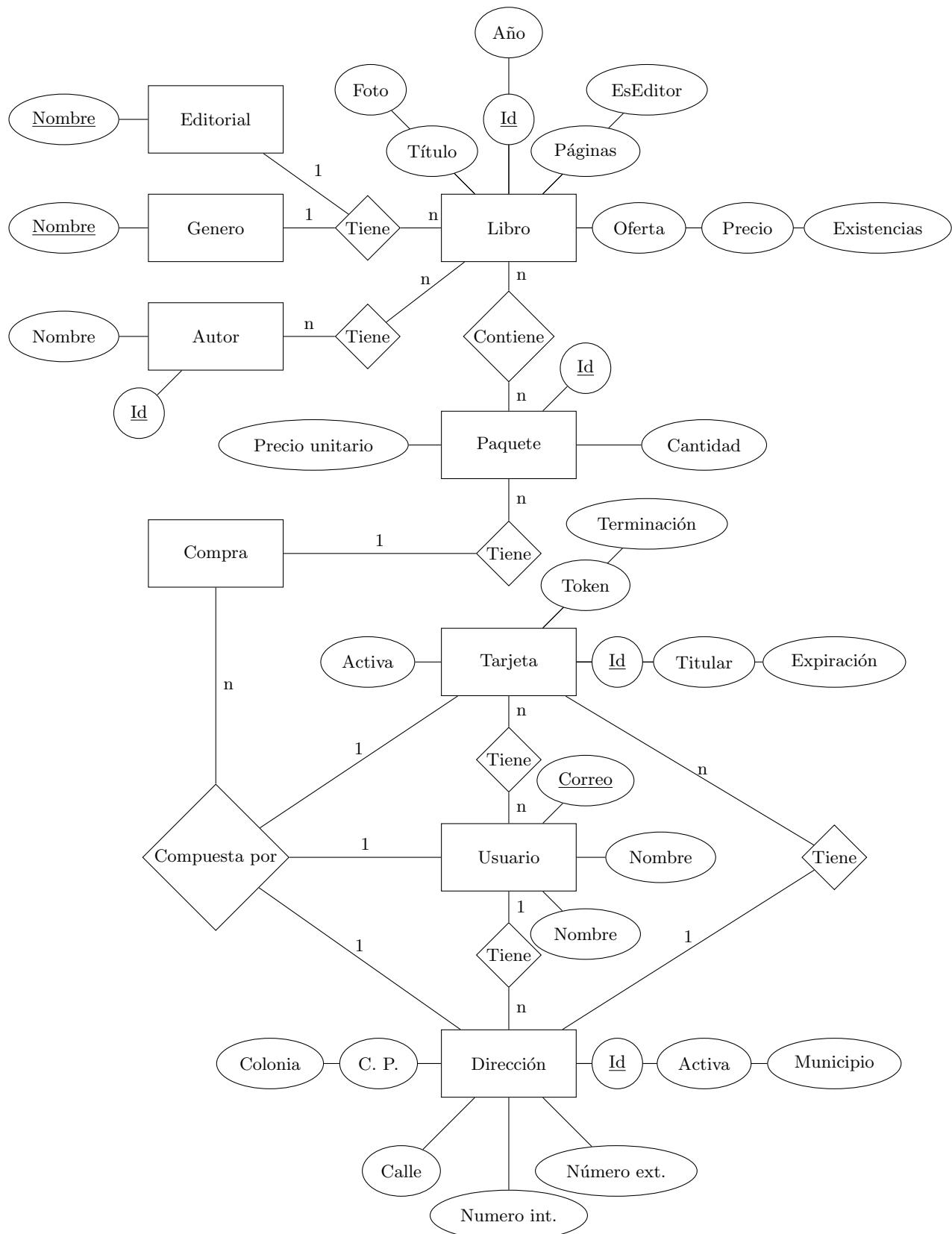


Figura 8.15: Diagrama entidad-relación para la tienda en línea.

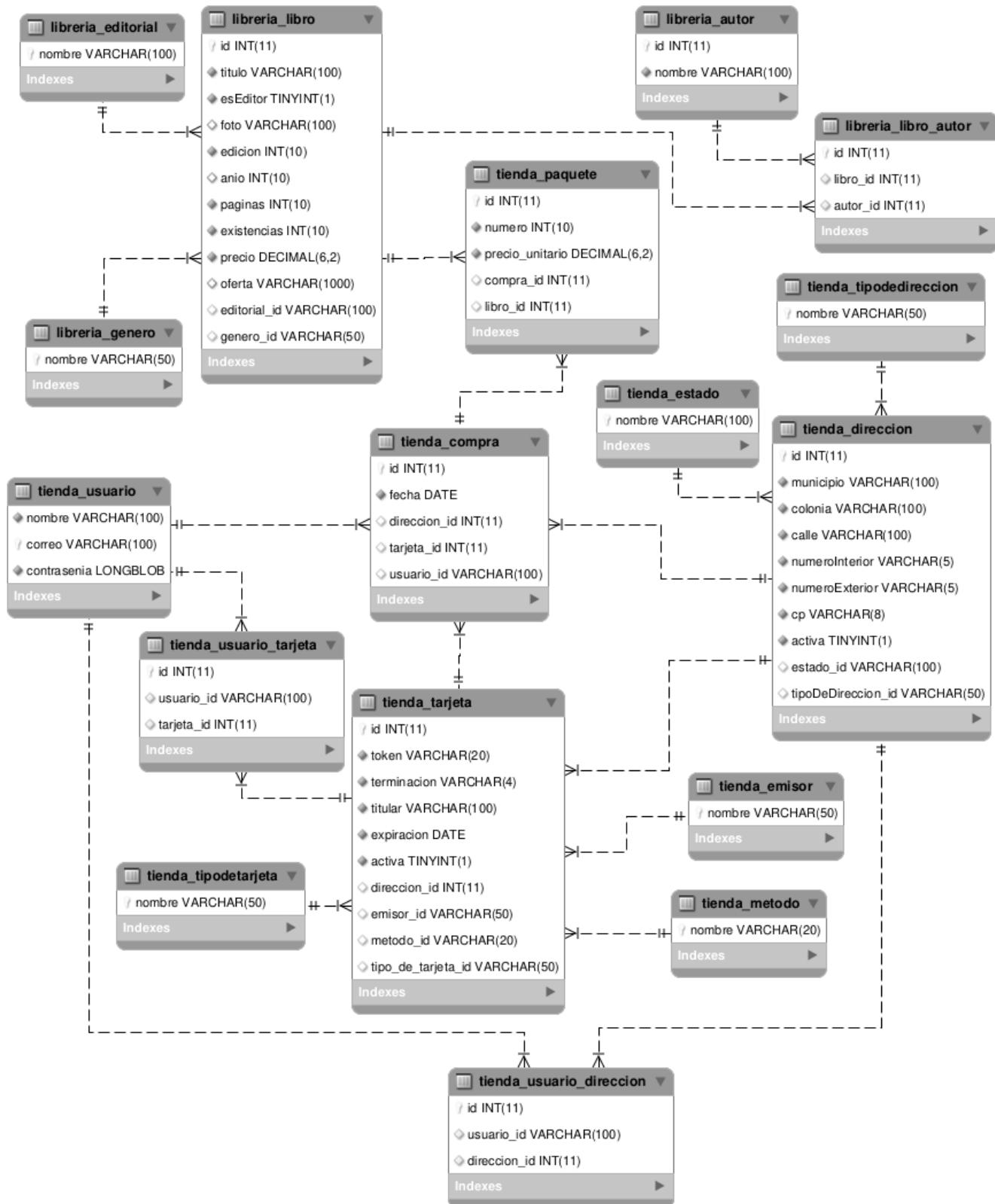


Figura 8.16: Diagrama relacional de la base de datos.

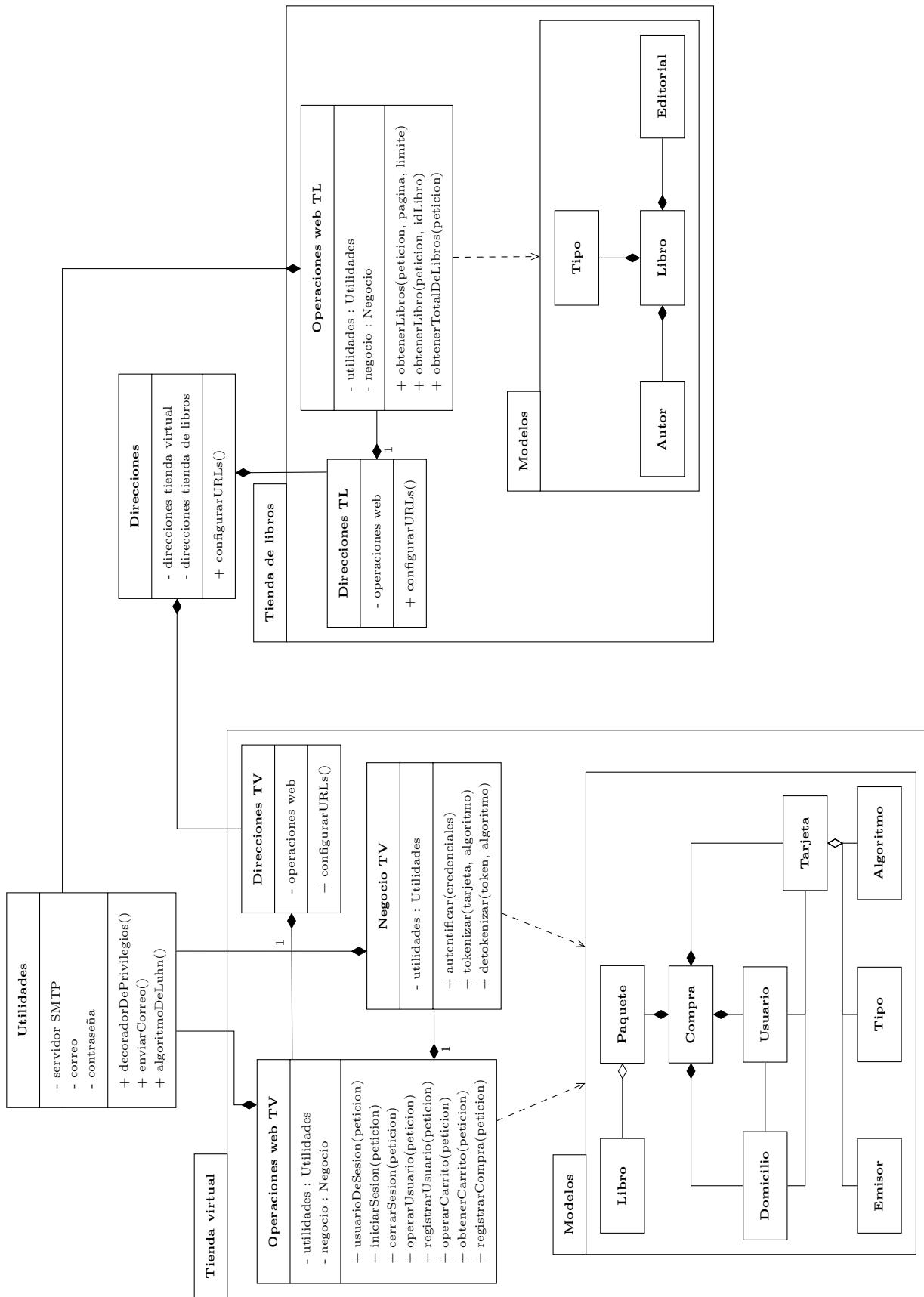


Figura 8.17: Diagrama de clases la tienda en línea.

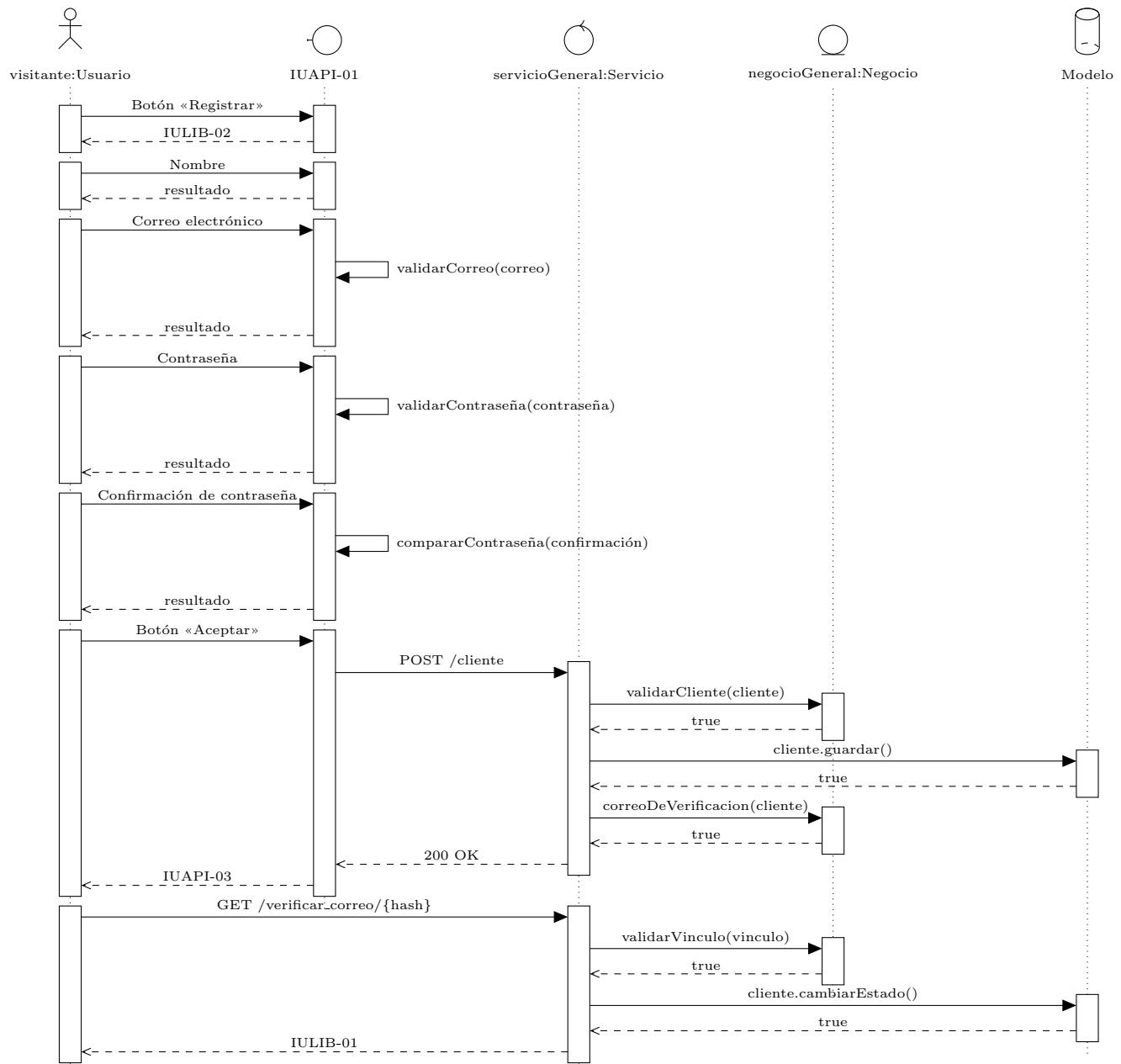


Figura 8.18: Diagrama de secuencia para registrar a un cliente.

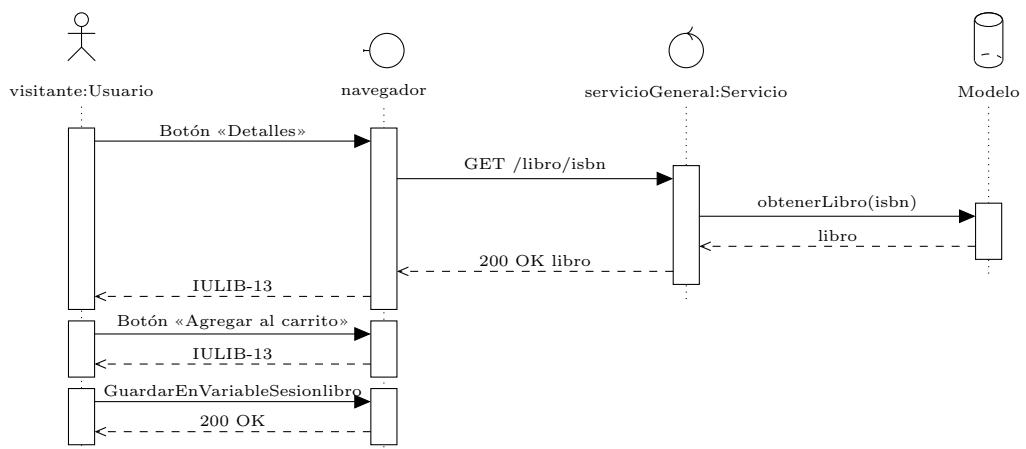


Figura 8.19: Diagrama de secuencia para agregar un libro al carrito.

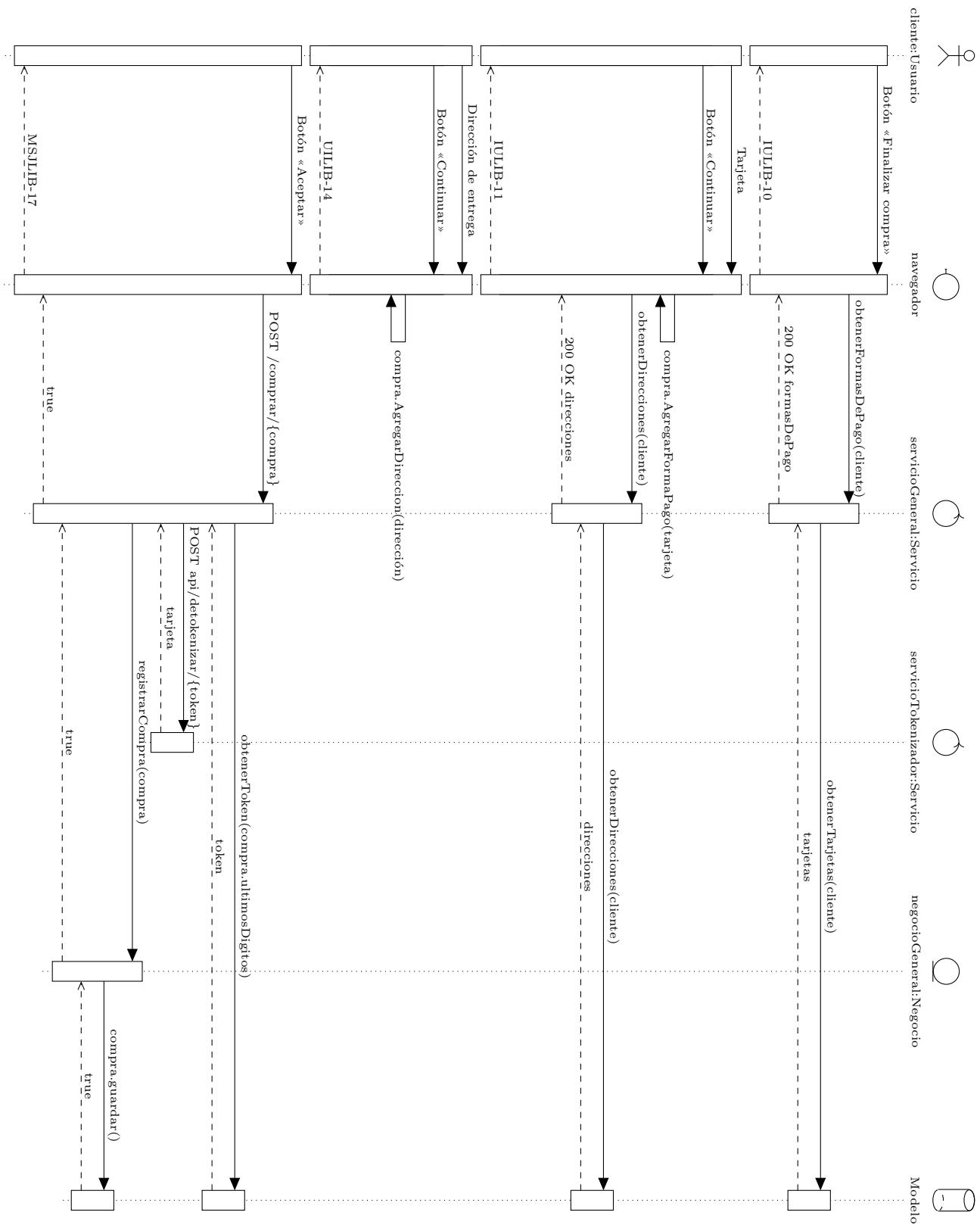


Figura 8.20: Diagrama de secuencia para realizar una compra.

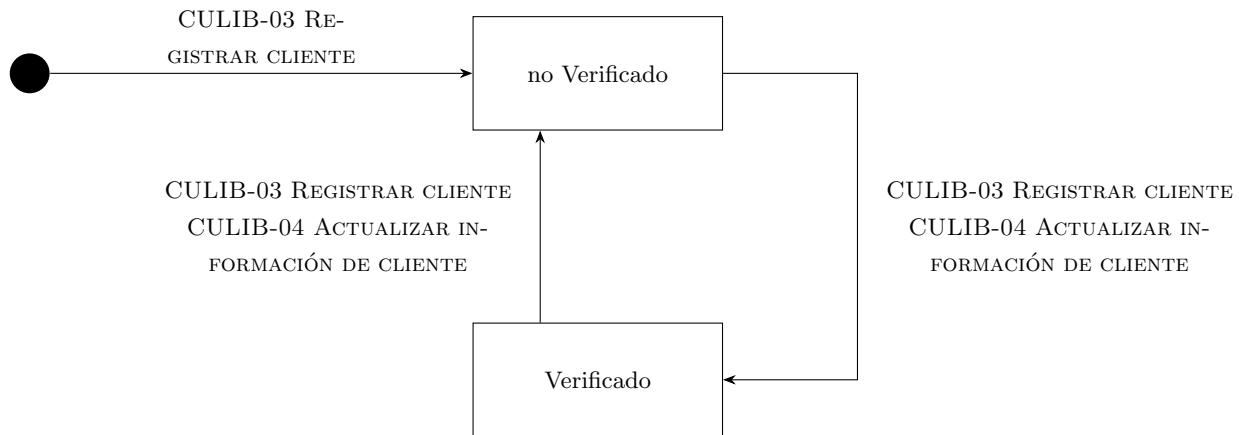


Figura 8.21: Diagrama de estados para un cliente.

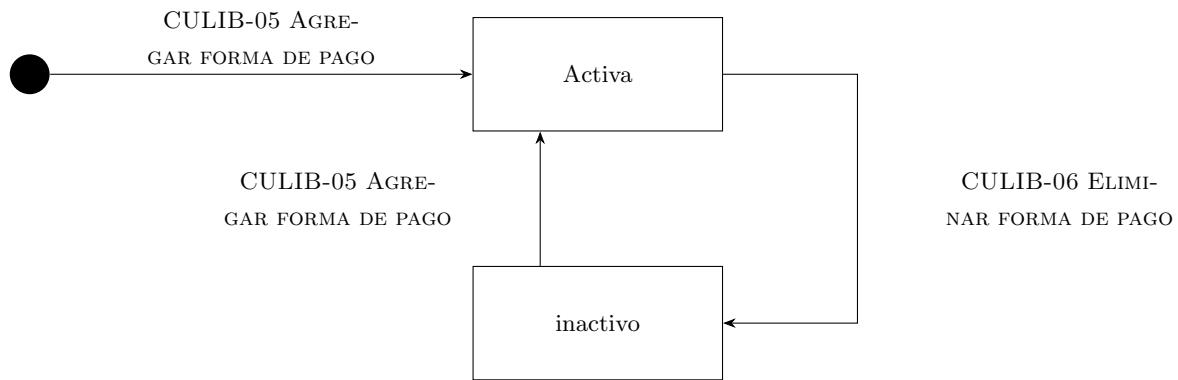


Figura 8.22: Diagrama de estados para una tarjeta.

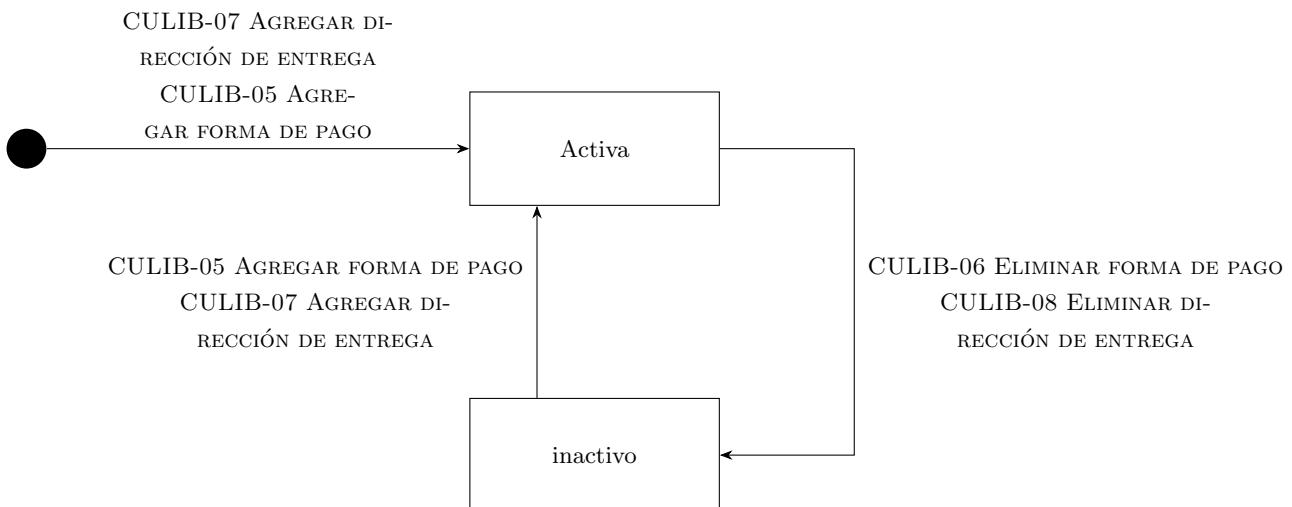


Figura 8.23: Diagrama de estados para una dirección.

Capítulo 9

Implementación de tienda en línea

*«You don't understand anything until you
learn it more than one way.»*

MARVIN MINSKY.

En esta sección se presentan los aspectos más relevantes de la implementación. Las tecnologías utilizadas son las mismas que en la aplicación web del sistema tokenizador.

9.1. Aspectos relevanates de la implementación

Al estar desarrollada sobre las mismas tecnologías, la implementación de la tienda en línea es bastante similar a la del sistema tokenizador. Por esta razón, el enfoque para mostrar ejemplos de código es distinto al de la sección 7.2 (los aspectos relevantes de la implementación del sistema tokenizador), en donde se mostró un ejemplo de cada módulo del programa, tanto en el servidor como en el cliente. Aquí se muestran los códigos fuente relacionados con la implementación de un solo caso de uso: CULIB-11 COMPRAR.

El primer paso consiste en la acción de inicio del usuario: el cliente presiona el botón *Finalizar compra* en IULIB-10 PÁGINA DE FINALIZAR UNA COMPRA. En el código 32 se muestra el fragmento de HYPERTEXT MARKUP LANGUAGE (HTML) correspondiente a ese botón. Como se puede observar en la línea 122, al presionar el botón se llama a la función `finalizarCompra`. Esta función se muestra en el código 33, que es el controlador asociado a la ventana del carrito.

```
107      <td md-cell style="text-align: right">
108          <p>
109              {{carrito.total | currency}}
110          </p>
111      </td>
112      <td md-cell></td>
113  </tr>
114  </tbody>
115 </table>
116 </md-table-container>
117 <div
118     layout="row"
119     layout-align="center">
120     <md-button
121         id="botonFinalizarCompra"
122         ng-click="finalizarCompra($event)"
123         class="md-primary md-raised">
124             Finalizar compra
125     </md-button>
126 </div>
127 </div>
```

Código fuente 32: Botón *Finalizar compra*, en pantalla de carrito.

TIENDA/ARCHIVOS_WEB/HTML/CARRITO.HTML

El siguiente paso consiste en mostrar la interfaz IULIB-11 PÁGINA DE SELECCIÓN DE LA FORMA DE PAGO PARA UNA COMPRA. Esto se logra con la llamada de la función del código 33. Lo primero que hace esta función es verificar una de las condiciones de inicio del caso CULIB-01 INICIAR SESIÓN: si el usuario presiona *Finalizar compra* sin haber iniciado sesión antes, se le redirige a la pantalla IULIB-03 FORMULARIO DE INICIO DE SESIÓN. Si el usuario ya inició sesión, se abre la ventana emergente con el

```
24   $scope.finalizarCompra = function ($event) {
25     if ($scope.usuario == undefined) {
26       $scope.iniciarSesion(undefined);
27       /* TODO: que iniciarSesion regrese una promesa: al terminar su flujo,
28        * se hace una llamada recursiva a esta función. */
29     } else {
30       $mdDialog.show({
31         parent: angular.element(document.body),
32         targetEvent: $event,
33         templateUrl: '/estaticos/html/ventanas/finalizar_compra.ventana.html',
34         controller: 'controladorFinalizarCompra',
35         focusOnOpen: false,
36         locals: {
37           'libros': $scope.carrito.libros,
38           'tarjetas': $scope.tarjetas,
39           'agregarMetodoDePago': $scope.agregarMetodoDePago,
40           'direcciones': $scope.direcciones,
41           'agregarDireccionDeEntrega': $scope.agregarDireccionDeEntrega
42         }
43       }).then(function (respuesta) {
44         if (respuesta != undefined) {
45           $scope.reiniciarCarrito();
46           $location.path('/');
47         }
48       });
49     }
50   };

```

Código fuente 33: Función de finalización de compra.

TIENDA/ARCHIVOS_WEB/JSCONTROLADORES/CARRITO.CONTROLADOR.JS

proceso de finalización de compra. El controlador asociado recibe, entre otras cosas, un objeto con los libros del carrito, un arreglo con las tarjetas asociadas al usuario (paso número dos del caso de uso) y un arreglo con las direcciones de entrega del usuario.

El controlador de la ventana del proceso de finalización de compra (`controladorFinalizarCompra`) le pide al servidor el HTML de la ventana a mostrar (`finalizar_compra.ventana.html`). En el código fuente 34 se muestra la secuencia de inicio de *javascript* ejecutada al momento de cargar la ventana. La variable `$scope.secuencia` controla las interfaces mostradas al usuario:

1. IULIB-11 PÁGINA DE SELECCIÓN DE LA FORMA DE PAGO PARA UNA COMPRA.
2. IULIB-12 PÁGINA DE SELECCIÓN DE LA DIRECCIÓN DE ENTREGA.
3. IULIB-13 PÁGINA DE RESUMEN DE COMPRA

El objeto `temporal` guarda la forma de pago y la dirección seleccionadas por el usuario. El objeto `libros` guarda la selección de libros del usuario. En el último fragmento del código 34 se puede observar una de las condiciones de inicio del caso de uso CULIB-05 AGREGAR FORMA DE PAGO: si el cliente no tiene ninguna forma de pago asociada, se muestra la interfaz IULIB-08 FORMULARIO DE INFORMACIÓN BANCARIA.

```

77  /* Secuencia de inicio. *****/
78
79  $scope.secuencia = 1;
80  $scope.temporal = {};
81  $scope.temporal.direccion = 0;
82  $scope.temporal.tarjeta = 0;
83
84  $scope.libros = libros;
85  $scope.precioTotal = 0;
86  for (i = 0; i < libros.length; i++) {
87      $scope.precioTotal += libros[i].cantidad * libros[i].precio
88  }
89
90  $scope.tarjetas = tarjetas;
91  $scope.agregarMetodoDePago = agregarMetodoDePago;
92
93  /* TODO:
94   * Hacer que «agregarMetodoDePago» regrese una promes; al terminar
95   * se tiene que seleccionar el nuevo método. */
96
97  $scope.direcciones = direcciones;
98  $scope.agregarDireccionDeEntrega = agregarDireccionDeEntrega;
99
100 /* TODO:
101  * ¿Qué pasa si el usuario le dá «cancelar» a la ventana para agregar
102  * método de pago? Se tiene que implementar lo de las promesas para
103  * identificar ese caso y cerrar esta ventana (sin métodos de pago no se
104  * debe poder continuar). */
105
106 if ($scope.tarjetas.length == 0) {
107     $scope.agregarMetodoDePago(undefined);
108 } else {
109     $scope.temporal.tarjeta = $scope.tarjetas[0];
110 }
```

Código fuente 34: Secuencia de inicio de controlador de finalización de compra.

TIENDA/ARCHIVOS_WEB/JS/CONTROLADORES/SECUNDARIOS/FINALIZAR_COMPRA.CONTROLADOR.JS

El siguiente paso consiste en la selección de la forma de pago. En el código fuente 35 se muestra el fragmento de la ventana en donde se encuentran los botones radiales con las formas de pago asociadas al cliente en sesión. También se muestra el botón para agregar una nueva forma de pago. Esto último representa otra condición de inicio del caso de uso CULIB-05 AGREGAR FORMA DE PAGO: si el cliente presiona el botón, se muestra la interfaz IULIB-08 FORMULARIO DE INFORMACIÓN BANCARIA.

```
22 <div
23   ng-if="secuencia == 1"
24   layout="column"
25   layout-align="center center">
26   <h3>
27     Selecciona el método de pago
28   </h3>
29   <md-divider></md-divider>
30   <md-radio-group ng-model="temporal.tarjeta">
31     <div
32       ng-repeat='tarjeta in tarjetas'
33       class="row">
34       <div flex
35         layout='row'
36         layout-padding
37         layout-align="start center">
38         <md-radio-button
39           flex
40           ng-value="tarjeta"
41           class="md-accent">
42             {{tarjeta.fields.titular}} -
43             <em>
44               XXXX XXXX XXXX {{tarjeta.fields.terminacion}}
45             </em> -
46             ({{tarjeta.fields.emisor}})
47           </md-radio-button>
48         </div>
49       </div>
50     </md-radio-group>
51     <md-button
52       ng-click="agregarMetodoDePago($event)"
53       class="md-primary">
54       Agregar método de pago
55     </md-button>
56   </div>
```

Código fuente 35: Selección de forma de pago.

TIENDA/ARCHIVOS_WEB/HTML/VENTANAS/FINALIZAR_COMPRA.VENTANA.HTML

Después de seleccionar la forma de pago, el cliente debe de presionar el botón *Continuar*. Este botón se muestra en el fragmento del código fuente 36; al ser presionado se llama a la función **continuar** (código fuente 37). Se trata del primer caso del condicional, en donde se muestra la interfaz IULIB-12 PÁGINA DE SELECCIÓN DE LA DIRECCIÓN DE ENTREGA y se verifica una de las condiciones de inicio del caso de uso CULIB-07 AGREGAR DIRECCIÓN DE ENTREGA: en caso de que el cliente no tenga direcciones asociadas, se muestra la ventana IULIB-09 FORMULARIO DE DIRECCIÓN DE ENTREGA.

En los códigos fuente 36 y 37 se muestra la trayectoria alternativa de cancelación. Si el cliente presiona

```
114  </md-dialog-content>
115  <md-dialog-actions>
116    <md-button
117      ng-click="cancelar()"
118      class="md-primary">
119      Cancelar
120    </md-button>
121    <md-button
122      ng-if="secuencia != 1"
123      ng-click="regresar()"
124      class="md-primary">
125      Regresar
126    </md-button>
127    <md-button
128      id="botonContinuarFinalizacionDeCompra"
129      ng-click="continuar($event)"
130      class="md-raised md-primary">
131      Continuar
132    </md-button>
133  </md-dialog-actions>
134 </md-dialog>
```

Código fuente 36: Acciones de ventana de finalización de compra.

[TIENDA/ARCHIVOS_WEB/HTML/VENTANAS/FINALIZAR_COMPRA.VENTANA.HTML](#)

el botón *Cancelar* se regresa a la interfaz IULIB-10 PÁGINA DE FINALIZAR UNA COMPRA.

El siguiente paso consiste en la selección de la dirección de entrega. En el código fuente 38 se muestra el fragmento de la ventana en donde se encuentran los botones radiales con las direcciones asociadas. También se muestra el botón para agregar una nueva dirección. Esto último es una condición de inicio más del caso de uso CULIB-07 AGREGAR DIRECCIÓN DE ENTREGA: si el cliente presiona el botón, se muestra la interfaz IULIB-09 FORMULARIO DE DIRECCIÓN DE ENTREGA.

En la línea 77 del código 38 se puede ver el uso de un componente propio: *ego-direccion*. Los componentes de *Angular* permiten reutilizar código a lo largo de una aplicación web [59]. En este caso, el componente de la dirección está definiendo el formato con el que se muestran las direcciones: un cambio en este formato afecta a todos los puntos de la aplicación que muestran direcciones. En el código fuente 39, se muestra el registro del componente con la aplicación; y en el código 40, se muestra el contenido del componente.

Continuando con el caso de uso, después de seleccionar la dirección de entrega, el siguiente paso consiste en presionar el botón de *Continuar* (código fuente 36), que ejecuta el segundo bloque del condicional del código 37. En este caso, simplemente se muestra la interfaz IULIB-13 PÁGINA DE RESUMEN DE COMPRA. En el código fuente 41, se muestra el HTML de esta interfaz.

El siguiente paso es volver a presionar el botón *Continuar*. En este caso, se pasa al tercer bloque del condicional de la función del código fuente 37. Aquí se llama a la función *api.registrarCompra*, que es la encargada de comunicarse con el programa servidor. En el código fuente 42, se muestra un fragmento de

```
36  $scope.cancelar = function () {
37    $mdDialog.hide();
38  };
39
40  $scope_continuar = function ($event) {
41    if ($scope.secuencia == 1) {
42      $scope.secuencia = 2;
43      if ($scope.direcciones.length == 0) {
44        $scope.agregarDireccionDeEntrega(undefined);
45      } else {
46        $scope.temporal.direccion = $scope.direcciones[0];
47      }
48    } else if ($scope.secuencia == 2) {
49      $scope.secuencia = 3;
50    } else if ($scope.secuencia == 3) {
51      api.registrarCompra({
52        'tarjeta': $scope.temporal.tarjeta.pk,
53        'direccion': $scope.temporal.direccion.pk})
54      .then(function (respuesta) {
55        var aviso = $mdDialog.alert()
56          .title('Registro exitoso')
57          .textContent('Su compra ha sido registrada con el número de tarjeta: '
58            + respuesta.data.substring(0, 4) + ' '
59            + respuesta.data.substring(4, 8) + ' '
60            + respuesta.data.substring(8, 12) + ' '
61            + respuesta.data.substring(12, 16) + '.')
62          .ariaLabel('Registro exitoso')
63          .targetEvent($event)
64          .ok('Aceptar')
65          .multiple(true);
66        $mdDialog.show(aviso).then(function (respuesta) {
67          $mdDialog.hide("todo en orden");
68        });
69      });
70    }
71  };
72
73  $scope_regresar = function () {
74    $scope.secuencia--;
75  };

```

Código fuente 37: Funciones de controlador de finalización de compra.

TIENDA/ARCHIVOS_WEB/JS/CONTROLADORES/SECUNDARIOS/FINALIZAR_COMPRA.CONTROLADOR.JS

```
57 <div  
58   ng-if="secuencia == 2"  
59   layout="column"  
60   layout-align="center center">  
61     <h3>  
62       Selecciona la dirección de entrega  
63     </h3>  
64     <md-divider></md-divider>  
65     <md-radio-group ng-model="temporal.direccion">  
66       <div  
67         ng-repeat='direccion in direcciones'  
68         class="row">  
69           <div flex  
70             layout='row'  
71             layout-padding  
72             layout-align="start center">  
73             <md-radio-button  
74               flex  
75               ng-value="direccion"  
76               class="md-accent">  
77               <ego-direccion informacion="direccion"></ego-direccion>  
78             </md-radio-button>  
79           </div>  
80         </div>  
81       </md-radio-group>  
82       <md-button  
83         ng-click="agregarDireccionDeEntrega($event)"  
84         class="md-primary">  
85         Agregar dirección de entrega  
86       </md-button>
```

Código fuente 38: Selección de dirección de entrega.

TIENDA/ARCHIVOS_WEB/HTML/VENTANAS/FINALIZAR_COMPRA.VENTANA.HTML

```
1 /*  
2  * Componente para mostrar en una cadena de una dirección.  
3  * Tienda en línea.  
4  * Proyecto Lovelace.  
5  */  
6  
7 tienda.component('egoDireccion', {  
8   restrict: 'E',  
9   templateUrl: '/estaticos/html/componentes/direccion.componente.html',  
10  bindings: {  
11    informacionDeDireccion: '=informacion'  
12  }  
13});
```

Código fuente 39: Registro de componente de dirección.

TIENDA/ARCHIVOS_WEB/JS/COMPONENTES/DIRECCION.COMPONENTE.JS

```

1 <!--
2   Plantilla para directiva de dirección.
3   Tienda en línea.
4   Proyecto Lovelace.
5 -->
6
7 {{ $ctrl.informacionDeDireccion.fields.calle}},
8 {{ $ctrl.informacionDeDireccion.fields.numeroInterior}} -
9 {{ $ctrl.informacionDeDireccion.fields.numeroExterior}},
10 {{ $ctrl.informacionDeDireccion.fields.colonia}},
11 {{ $ctrl.informacionDeDireccion.fields.municipio}},
12 {{ $ctrl.informacionDeDireccion.fields.estado}},
13 C.P. {{ $ctrl.informacionDeDireccion.fields.cp}}.

```

Código fuente 40: Contenido de componente de dirección.

TIENDA/ARCHIVOS_WEB/HTML/COMPONENTES/DIRECCION.COMPONENTE.HTML

```

88 <div
89   ng-if="secuencia == 3"
90   layout="column"
91   layout-align="center center">
92   <h3>
93     Resumen de compra
94   </h3>
95   <p>
96     <em>Forma de pago:</em> <br>
97     {{temporal.tarjeta.fields.titular}} -
98     XXXX XXXX XXXX {{temporal.tarjeta.fields.terminacion}}
99     ({{temporal.tarjeta.fields.emisor}})
100  </p>
101  <p>
102    <em>Punto de entrega:</em> <br>
103    <ego-direccion informacion="temporal.direccion"></ego-direccion>
104  </p>
105  <p>
106    <em>Libros:</em> <br>
107    <em ng-repeat='libro in libros'>
108      {{libro.cantidad}} - {{libro.titulo}},
109      {{libro.autor[0]}}. {{libro.precio | currency}}<br>
110    </em>
111    <em>Precio total:</em> {{precioTotal | currency}} <br>
112  </p>
113  </div>
114 </md-dialog-content>

```

Código fuente 41: Resumen de compra.

TIENDA/ARCHIVOS_WEB/HTML/VENTANAS/FINALIZAR_COMPRA.VENTANA.HTML

las funciones que se comunican con el servidor. En el caso de `registrarCompra`, se hace una petición POST a la UNIFORM RESOURCE LOCATOR (URL) `/api/tienda/compra`, con un objeto JAVASCRIPT OBJECT NOTATION (JSON) que representa la compra.

```
54 API.obtenerTotalDeLibros = function () {
55     return $http.get(RUTA_BASE + RUTA_LIBRERIA + '/total_de_libros');
56 };
57
58 /* Operaciones de carrito. *****/
59
60 API.guardarCarrito = function (carrito) {
61     return $http.post(RUTA_BASE + RUTA_TIENDA + '/carrito', carrito);
62 };
63
64 API.obtenerCarrito = function () {
65     return $http.get(RUTA_BASE + RUTA_TIENDA + '/carrito');
66 };
67
68 API.registrarCompra = function (compra) {
69     return $http.post(RUTA_BASE + RUTA_TIENDA + '/compra', compra);
70 };
71
72 /* Operaciones sobre tarjetas *****/
73
74 API.obtenerTarjetas = function () {
75     return $http.get(RUTA_BASE + RUTA_TIENDA + '/tarjetas');
76 };
```

Código fuente 42: Fragmento de funciones de comunicación con el servidor.

[TIENDA/ARCHIVOS_WEB/JS/SERVICIOS/API.SERVICIO.JS](#)

En el código fuente 43, se muestra un fragmento de la configuración global de direcciones de la aplicación de la tienda. En el caso de la petición para registrar una compra, se pasa a la configuración de direcciones de la aplicación llamada *tienda*. En el código fuente 44, se muestra un fragmento de estas configuraciones. En el caso de la petición de registro de compra (línea 32), se llama a la función `funciones_web.registrarCompra`.

La función de registro de compra se muestra en el código 45. La primera parte de la función consiste en el registro en la base de datos de la compra (y las tablas intermedias asociadas); cada libro comprado se resta de las existencias del inventario. Por último, se llama a la función `negocio.detokenizar`, que es la encargada de comunicarse con el sistema tokenizador.

El código fuente 46, muestra la función de la capa de negocios para hacer detokenizaciones. En caso de que la respuesta del sistema tokenizador sea un código de respuesta 200, el flujo del caso de uso continúa por la trayectoria principal: se muestra el mensaje MSJLIB-17 COMPRA REGISTRADA.

```

1 """
2 direcciones.py Configuración de urls del sistema.
3 Tienda en linea.
4 Proyecto Lovelace.
5
6 Documentación asociada:
7 https://docs.djangoproject.com/en/2.0/topics/http/urls/
8 """
9
10 import django
11 import tienda.general as general
12
13 urlpatterns = [
14
15     # Operaciones de módulo general
16     django.urls.path('api/tienda/',
17         django.urls.include(
18             'tienda.tienda.direcciones')),
19
20     # Operaciones de módulo general
21     django.urls.path('api/libreria/',
22         django.urls.include(
23             'tienda.libreria.direcciones')),

```

Código fuente 43: Configuración global de direcciones.

TIENDA/DIRECCIONES.PY

```

21 # Operaciones de cliente
22 django.urls.path('operar_usuario',
23     funciones_web.operarUsuario),
24 django.urls.path('verificar_correo/registro/<vinculo>',
25     funciones_web.verificarCorreoDeRegistro),
26 django.urls.path('verificar_correo/actualizacion/<vinculo>',
27     funciones_web.verificarCorreoDeActualizacion),
28
29 # Operaciones de carrito
30 django.urls.path('carrito',
31     funciones_web.operarCarrito),
32 django.urls.path('compra',
33     funciones_web.registrarCompra),
34
35 # Operaciones sobre tarjetas
36 django.urls.path('tarjetas',
37     funciones_web.obtenerTarjetas),
38 django.urls.path('direccion_de_tarjeta/<int:idDeDireccion>',
39     funciones_web.obtenerDireccionDeTarjeta),

```

Código fuente 44: Configuración de direcciones de aplicación *tienda*.

TIENDA/TIENDA/DIRECCIONES.PY

```
203 @utilidades.privilegiosRequeridos
204 def registrarCompra(peticion):
205     """Registra una compra del cliente en sesión."""
206
207     objetoDePeticion = json.loads(peticion.body)
208     carrito = json.loads(peticion.session['carrito'])
209     identificador = json.loads(peticion.session['usuario'])['pk']
210
211     # Registrar compra
212     compra = Compra(
213         fecha = datetime.datetime.now(),
214         tarjeta = Tarjeta.objects.get(pk = objetoDePeticion['tarjeta']),
215         usuario = Usuario.objects.get(pk = identificador),
216         direccion = Direccion.objects.get(pk = objetoDePeticion['direccion']))
217     compra.save()
218
219     tarjeta = Tarjeta.objects.get(pk = objetoDePeticion['tarjeta'])
220
221     # Registrar libros de compra
222     for libro in carrito['libros']:
223         paquete = Paquete(
224             libro = libreria.models.libro.Libro.objects.get(pk = libro['pk']),
225             compra = compra,
226             numero = libro['cantidad'],
227             precio_unitario = libro['precio'])
228         paquete.save()
229
230         # Restar de existencias
231         paquete.libro.existencias -= paquete.numero;
232         paquete.libro.save()
233
234     del peticion.session['carrito']
235
236     try:
237         numeroDeTarjeta = negocio.detokenizar(tarjeta.token, str(tarjeta.metodo))
238         return django.http.HttpResponse(numeroDeTarjeta)
239     except Exception as error:
240         print(traceback.format_exc())
241         return django.http.HttpResponse()
```

Código fuente 45: Función de registro de compra.

TIENDA/TIENDA/FUNCIONES_WEB.PY

```
168 def detokenizar (token, metodo):
169     """Realiza una operación de detokenización.
170
171     Hace un post al sistema tokenizador.
172     En caso de error, se levanta una excepción de error de sistema.
173     """
174     peticion = requests.post(
175         configuraciones.SISTEMA_TOKENIZADOR
176         + '/api/programa_tokenizador/detokenizar',
177         auth = (configuraciones.USUARIO_ST, configuraciones.CONTRASENIA_ST),
178         data = json.dumps({'token': token, 'metodo': metodo}))
179     if peticion.status_code != 200:
180         raise SystemError(
181             'Error en detokenización: {}'.format(peticion.status_code))
182     return peticion.text.replace('\n', '')
```

Código fuente 46: Función de comunicación con sistema tokenizador: operación de detokenización.

TIENDA/TIENDA/NEGOCIO.PY

Capítulo 10

Conclusiones

*«Reserve your right to think, for even to
think wrongly is better than not to think
at all.»*

HYPATIA.

Como se planteó al inicio, este trabajo fue desarrollado en tres grandes módulos:

Programa generador de tokens Aquí se analizaron e implementaron cinco algoritmos tokenizadores: FFX, BPS, TKR, AHR y DRBG.

Servicio de tokenización Se encarga de proveer una interfaz de comunicación entre el programa generador de tokens y los posibles clientes; además, incluye una aplicación web que permite gestionar a los usuarios.

Caso de prueba: tienda de libros en línea Se implementaron, principalmente, las partes que involucran comunicación con el servicio de tokenización (agregar método de pago y simulación de una compra).

Sobre el desempeño de los algoritmos, se concluye que los reversibles (FFX y BPS) son considerablemente más rápidos que los tres irreversibles (TKR, AHR y DRBG). También es posible observar que, para los métodos irreversibles, el proceso de detokenización es mucho más rápido que la generación de tokens. Estos dos resultados dejan ver un poco la carga de las operaciones en los métodos irreversibles: la tokenización involucra una consulta a la base de datos, la generación del TOKEN y una inserción; la detokenización solamente es una consulta.

El que FFX y BPS sean más rápidos puede resultar un poco contraintuitivo, pues la generación de tokens reversibles involucra más operaciones; cuando en la medición de tiempos se excluye el acceso a la base de datos, el algoritmo más veloz es DRBG, seguido de cerca por TKR y AHR; los dos reversibles terminan siendo los más lentos.

Además de los tiempos de ejecución, también es importante señalar que los irreversibles, al operar como funciones de un solo sentido, son más seguros que los reversibles: un atacante con acceso a la llave de cifrado puede obtener el número de tarjeta correspondiente si se trata de un método reversible, mientras que con un método irreversible necesita también acceso a la base de datos.

También es posible concluir que la denominación *no criptográficos*, de la clasificación del PCI DSS resulta totalmente confusa, pues en realidad todos los métodos conocidos que caen en esa categoría utilizan primitivas criptográficas. La segunda categoría (irreversibles) carece de utilidad para aplicaciones que procesan pagos con tarjetas de crédito, pues la capacidad de regresar al número de tarjeta a partir de su token es uno de los requerimientos principales para los sistemas tokenizadores.

Respecto a los objetivos planteados en la sección 1.1, podemos concluir que se satisficieron los tres objetivos específicos: revisar diversos algoritmos para la generación de TOKENS (corresponde al módulo de *Programa generador de tokens*); diseñar e implementar un servicio web que proporcione el servicio de generación de tokens (corresponde al módulo de *Servicio de tokenización*) y, finalmente, implementar una

tienda que use el servicio de tokenización (corresponde al módulo de *Caso de prueba*). También se cumplió el objetivo general, pues se implementaron algoritmos criptográficos y no criptográficos¹.

¹Tomando en cuenta la clasificación propuesta por el PAYMENT CARD INDUSTRY (PCI) SECURITY STANDARD COUNCIL (SSC), descrita en la sección 3.1.

Glosario

«The most important thing in a programming language is the name. A language will not succeed without a good name. I have recently invented a very good name and now I am looking for a suitable language.»

DONALD KNUTH.

Glosario de términos de computación, criptográficos y matemáticos. Las principales fuentes bibliográficas usadas son [19] y [60]; en caso de tratarse de una fuente distinta, se indica en la entrada en particular.

1. Acoplamiento

(*Coupling*) Medida de la fuerza de asociación establecida por la conexión de dos módulos dentro de un sistema. Un acoplamiento fuerte complica a un sistema, dado que es más difícil de entender, cambiar o corregir. La complejidad de un programa se puede reducir buscando el menor acoplamiento posible entre módulos [61]. 1, 70, 90, 335

2. Autenticación de origen

Tipo de autenticación donde se corrobora que una entidad es la fuente original de la creación de un conjunto de datos en un tiempo específico. Por definición, la *autenticación de origen* incluye la integridad de datos, pues cuando se modifican los datos, se tiene una nueva fuente. 1, 312, véase también INTEGRIDAD DE DATOS.

3. Autenticación multifactor

(*Multi-factor authentication*) Método de autenticación que requiere al menos dos métodos (independientes entre sí) para identificar al usuario. 1, 7, 60, 67, 167

4. Autenticación mutua

(*Mutual authentication*) Autenticación en la cual cada una de las partes identifica a la otra. 1, 60

5. Biyección

Dicho de las funciones que son inyectivas y suprayectivas al mismo tiempo; en otras palabras, que todos los elementos del conjunto de salida tengan una imagen distinta en el conjunto de llegada y a cada elemento del conjunto de llegada le corresponde un elemento del conjunto de salida. 1, 17, véase también INYECTIVA, SUPRAYECTIVA, FUNCIÓN y IMAGEN.

6. Cifrado con prefijo

Técnica de ordenamiento pseudoaleatorio que consiste en seguir el orden dado por el texto cifrado de todos los elementos a ordenar. 1, 31

7. Cifrado de caminata cíclica

(*Cycle-walking cipher*) Método diseñado para cifrar mensajes de un espacio M utilizando un algoritmo de cifrado por bloques que actúa en un espacio $M' \supset M$ y obtener textos cifrados que están en M al cifrar iterativamente hasta que el mensaje cifrado se encuentra en el dominio deseado. 1, 53, 103, 114

8. Cifrado iterativo

(*Iterated block cipher*) Cifrado de bloque que involucra la repetición secuencial de una función interna llamada función de ronda. Los parámetros incluyen el número de rondas, el tamaño de bloque y el tamaño de llave. 1, 15, véase también RONDA.

9. Circuito booleano

(*Boolean circuit*) Modelo matemático definido en términos de compuertas lógicas digitales (AND, OR, NOT, etc.). 1

10. Codominio

Una función mapea a los elementos de un conjunto A con elementos de un conjunto B ; A es el dominio y B es el *codominio*. 1, véase también FUNCIÓN y DOMINIO.

11. Combinación lineal

En álgebra lineal, una combinación lineal es la expresión resultante de la suma de n vectores v_i multiplicados por n escalares c_i de la siguiente forma: $\sum_{i=1}^n c_i v_i$. 1

12. Computacionalmente indistinguible

(*Computational indistinguishability*) Para un A tomado de algún conjunto de distribución y un circuito booleano C (con las suficientes entradas), p_C^A es la probabilidad de que la salida del circuito booleano C sea 1 para una entrada de A . Se dice que los conjuntos de distribución $\{A_k\}$ y $\{B_k\}$ son *computacionalmente indistinguibles* si para cualquier familia de circuitos de tamaño polinomial $C = \{C_k\}$, la función $e(k) = |p_{A_k}^{C_k} - p_{B_k}^{C_k}|$ es despreciable [62].

Otra forma de expresarlo, en un contexto más criptográfico, es como la incapacidad de un adversario de distinguir si la salida de una primitiva criptográfica es una permutación aleatoria o una permutación pseudoaleatoria: una permutación P es segura (en términos de un ataque de texto cifrado conocido) cuando es *computacionalmente indistinguible* de una permutación aleatoria. 1, 304, véase también FUNCIÓN, FUNCIÓN DESPRECIABLE, CONJUNTO DE DISTRIBUCIÓN y CIRCUITO BOOLEANO.

13. Computacionalmente no factible

(*Computationally infeasible*) Se dice que una tarea es *computacionalmente no factible* si su costo (medido en términos de espacio o de tiempo) es finito pero ridículamente grande. 1, 59

14. Conjunto de distribución

(*Distribution ensemble*) Un *conjunto de distribución* $\{A_k\}$ es una familia de medidas de probabilidad en $\{0, 1\}^*$ para la cual hay un polinomio q tal que las únicas cadenas de longitud mayor a $q(k)$ tienen una probabilidad distinta de cero en $\{A_k\}$ [62]. 1

15. Criptografía fuerte

(*Strong cryptography*) De acuerdo al PAYMENT CARD INDUSTRY (PCI) SECURITY STANDARD COUNCIL (SSC) (en [63]), es la criptografía basada en algoritmos probados y aceptados en la industria, junto con longitudes de llaves fuertes y buenas prácticas de administración de llaves. 1, 38, 60, 314

16. Criptología

Estudio de los sistemas, claves y lenguajes secretos u ocultos. 1, 11

17. Criptomonedas

(*Cryptocurrency*) Moneda digital que funciona con herramientas criptográficas modernas. La primera y más exitosa a la fecha, llamada *bitcoin*, fue propuesta en [15] por el epónimo *Satoshi Nakamoto*. La característica más importante de las *criptomonedas* es que no están respaldadas por alguna autoridad central [64]. 1, 7, 8

18. Dependencia lineal

Un conjunto de vectores es linealmente independiente si ninguno de ellos puede ser representado por una combinación lineal de los vectores restantes. Si los vectores no son linealmente independientes, se dice que existen dependencias lineales. 1, 298, véase también COMBINACIÓN LINEAL.

19. Distribución de probabilidad

Una distribución de probabilidad P en el conjunto de eventos S es una secuencia de números positivos p_1, p_2, \dots, p_n que sumados dan 1. Donde p_i se interpreta como la probabilidad de que el evento s_i ocurra. 1, 295

20. Distribución uniforme

Distribución de probabilidad en la que todos los elementos tienen la misma probabilidad ocurrencia. 1, 295, 307, véase también DISTRIBUCIÓN DE PROBABILIDAD.

21. Dominio

El *dominio* de una función $f(x)$ es el conjunto de valores para los cuales la función está definida. 1, véase también FUNCIÓN y CODOMINIO.

22. Entropía

Definida para una función de probabilidad de distribución discreta, mide cuánta información en promedio es requerida para identificar muestras aleatorias de esa distribución. 1, 15, 61, 282–285, 287, 291, 294, 302–304, véase también FUNCIÓN y DISTRIBUCIÓN DE PROBABILIDAD.

23. Equiprobable

Se dice que un conjunto de eventos es equiprobable cuando cada uno tiene la misma probabilidad de ocurrencia. 1, 51, 64

24. Estadísticamente independiente

Dicho de la ocurrencia de dos eventos E_1 y E_2 : si $P(E_1 \cap E_2) = P(E_1)P(E_2)$ entonces E_1 y E_2 son *estadísticamente independientes* entre sí. Es importante notar que si esto ocurre, entonces $P(E_1|E_2) = P(E_1)$ y $P(E_2|E_1) = P(E_2)$, es decir, la ocurrencia de uno no tiene ninguna influencia en las probabilidades de ocurrencia del otro. 1, 64, 65, véase también PROBABILIDAD CONDICIONAL.

25. Expresión regular

Forma algebraica de definir patrones. El conjunto de patrones que puede ser expresado mediante las expresiones regulares es exactamente el mismo que el conjunto de patrones que puede ser descrito por las transiciones de una máquina de estados finitos [65]. 1, 124, véase también MÁQUINA DE ESTADOS FINITOS.

26. Fuerza efectiva

Para un espacio de llaves K , su *fuerza efectiva* es $\log_2 |K|$ (el logaritmo base dos de su cardinalidad). 1, 63, 65, 314

27. Función

Regla entre dos conjuntos A y B de manera que a cada elemento del conjunto A le corresponda un único elemento del conjunto B . 1, véase también CODOMINIO y DOMINIO.

28. Función booleana

Son las funciones que mapean f a un valor del conjunto booleano 0, 1 o *verdadero* y *falso*. Formalmente, se define como $f : B^n \rightarrow B$, donde $B = \{0, 1\}$ y n un entero no negativo que corresponde al número de argumentos, o variables, que necesita la función. 1, 52, véase también FUNCIÓN.

29. Función despreciable

(*Negligible function*) Una función $e : N \rightarrow R$ es *despreciable* si, para todos los enteros positivos c , existe un entero N_c tal que para todo $x \geq N_c$, $|e(x)| \leq \frac{1}{x^c}$. Esto significa que $e(x)$ se desvanece más rápido que el inverso de cualquier polinomio [62]. 1, véase también FUNCIÓN.

30. Imagen

Suponga que se tiene $x \in X$ y $y \in Y$ tal que $f(x) = y$; se dice entonces que y es la *imagen* de x bajo f , o que x es preimagen de y . 1, véase también PREIMAGEN, FUNCIÓN, DOMINIO y CODOMINIO.

31. Integridad de datos

Propiedad en la que los datos no han sido alterados sin autorización desde que fueron creados, transmitidos o almacenados por una fuente autorizada. Operaciones que insertan, eliminan, modifican o reordenan bits invalidan la *integridad de los datos*. La *integridad de los datos* incluye que los datos estén completos y, cuando los datos son divididos en bloques, cada bloque debe cumplir con lo mencionado anteriormente.¹

32. Inyectiva

Una función $f : D_f \rightarrow C_f$ es *inyectiva* (o uno a uno) si a diferentes elementos del dominio le corresponden diferentes elementos del codominio; se cumple para dos valores cualesquiera $x_1, x_2 \in D_f$ que $x_1 \neq x_2 \implies f(x_1) \neq f(x_2)$.¹, véase también FUNCIÓN, DOMINIO, CODOMINIO, BIYECCIÓN y SUPRAYECTIVA.

33. Libreta de un solo uso

(*One-time pad*) Algoritmo de cifrado donde el texto en claro se combina con una llave secreta que es, al menos, de la longitud del mensaje. Si es utilizada e implementada correctamente, este algoritmo es indescifrable.¹

34. Material de llaves

(*keying material*) Conjunto de llaves criptográficas sin un formato específico.^{1, 304, 305, 307, 309}

35. Modo de operación

Construcción que permite extender la funcionalidad de un cifrado a bloques para operar sobre tamaños de información arbitrarios.^{1, 14, 24–28, 270, 273, 278, 279}

36. Máquina de estados finitos

Modelo idealizado de una máquina que opera en tiempos discretos. La operación completa de la máquina se define por un conjunto finito de estados junto con las reglas de cambios entre estados [66].

¹

37. Máquina de Turing

(*Turing machine*) Se considera como una cinta infinita dividida en casillas, cada una de las cuales contiene un símbolo. Sobre dicha cinta actúa un dispositivo que puede adoptar distintos estados y que, en cada instante, lee un símbolo de la casilla sobre la que está situado; dependiendo del símbolo leído y del estado en el que se encuentra, la máquina realiza las siguientes tres acciones: primero, pasa a un nuevo estado; segundo, imprime un símbolo en el lugar del que acaba de leer; y, tercero, se desplaza hacia la derecha, hacia la izquierda o se detiene. De forma más concreta, una máquina de Turing puede ser vista como una máquina de estados finitos con un método de almacenamiento asociado (la cinta infinita) [66]-[68].¹, véase también MÁQUINA DE ESTADOS FINITOS.

38. Nonce

Valor que varía con el tiempo y es improbable que se repita. Por ejemplo, puede ser un valor aleatorio generado para cada uso, una etiqueta de tiempo, un número de secuencia o una combinación de los tres. 1, 282–285

39. Oráculo

Oracle machine Se refiere a una máquina abstracta utilizada para estudiar problemas de decisión. Puede verse como una máquina de Turing con una caja negra (llamada *oráculo*) que puede resolver ciertos problemas de decisión u obtener el valor de una función en una sola operación. 1, 62, véase también MÁQUINA DE TURING.

40. Permutación

Sea S un conjunto finito de elementos. Una *permutación* p en S es una biyección de S a sí misma (i.e. $p : S \rightarrow S$). 1, 12, 63, 64, véase también BIYECCIÓN.

41. Preimagen

Suponga que se tiene $x \in X$ y $y \in Y$ tal que $f(x) = y$; se dice entonces que x es *preimagen* de y , o que y es la imagen de x bajo f . 1, 29, 290, véase también IMAGEN, FUNCIÓN, DOMINIO y CODOMINIO.

42. Primitiva criptográfica

(*Cryptographic primitive*) Algoritmos criptográficos que son usados con frecuencia para la construcción de protocolos de seguridad. En [19] se clasifican en tres categorías principales: de llave simétrica, de llave pública y sin llave. 1, 38, 40, 41, 50, 51, 54, 62, 278

43. Probabilidad condicional

Sean E_1 y E_2 dos eventos, con $P(E_2) \geq 0$. La *probabilidad condicional* se denota por $P(E_1|E_2)$, y es igual a

$$P(E_1|E_2) = \frac{P(E_1 \cap E_2)}{P(E_2)}$$

Esto mide la probabilidad de que ocurra E_1 sabiendo que ya ocurrió E_2 1

44. Prueba de componente

(*Component testing*) Pruebas de componentes hechas a partir de objetos que interactúan entre sí [69]. 1, 79

45. Prueba de unidad

(*Unit testing*) Proceso para probar componentes más simples de un programa de forma individual (funciones u objetos aislados) [69]. 1, 79

46. Registro de desplazamiento

Es un arreglo de *flip-flops* que se encarga de desplazar la información contenida en su estado actual, teniendo una entrada de un bit, para reemplazar el valor del primer *flip-flop* del arreglo. 1

47. Registro de desplazamiento con retroalimentación lineal

(*Linear-feedback shift register (LFSR)*) Es un registro de desplazamiento en el que su bit de entrada es una función lineal de su estado anterior. 1, 299, véase también REGISTRO DE DESPLAZAMIENTO.

48. Ronda

(*Round*) Bloque compuesto por un conjunto de operaciones que es ejecutado múltiples veces. Las *rondas* son definidas por el algoritmo de cifrado. 1, 15, 17, 19, 20

49. Semilla

(*Seed*) Cadena de bits que es utilizada como entrada para los los mecanismos DETERMINISTIC RANDOM BIT GENERATOR (DRBG). Determina una parte del estado interno del DRBG. VIII, 1, 282–285, 287, 288, 290, 294, 295, 304

50. Suprayectiva

Una función $f : D_f \rightarrow C_f$ es *suprayectiva* si todo elemento de su codominio C_f es imagen de por lo menos un elemento de su dominio D_f : $\forall b \in C_f \exists a \in D_f$ tal que $f(a) = b$. 1, véase también FUNCIÓN, DOMINIO, CODOMINIO, IMAGEN, BIYECCIÓN y INYECTIVA.

51. Tiempo polinomial

Se dice que una función computable o algoritmo es de *tiempo polinomial* cuando su complejidad está, en el peor de los casos, acotada por arriba por un polinomio sobre el tamaño de sus variables de entrada. Si se toma a f como una función computable, esta es de *tiempo polinomial* si $f \in O(n^k)$ donde $k \geq 1$. 1, 304

52. Token

Valor representativo que se usa en lugar de información valiosa. IV–VI, 1, 3–8, 10, 38–40, 50–52, 55, 58–70, 77, 89, 101, 103, 114, 124, 128, 133, 134, 138, 139, 146–149, 151, 165–168, 246, 270

53. Vector de inicialización

Cadena de bits de tamaño fijo que sirve como entrada a muchas primitivas criptográficas (p. ej. algunos modos de operación). Generalmente se requiere que sea generado de forma aleatoria. 1, 26, 28, 47, 306, 312, véase también MODO DE OPERACIÓN y PRIMITIVA CRIPTOGRÁFICA.

Siglas y acrónimos

«Por supuesto que la paz es el respeto al derecho ajeno, en eso todos estamos de acuerdo. En lo que nadie está de acuerdo es en cuál es el derecho ajeno.»

Instrucciones para vivir en México,
JORGE IBARGÜENGOITIA.

Criptográficos

AES Advanced Encryption Standard. III, 10, 13, 15, 20, 30, 42, 44, 61, 74, 77, 87, 90, 102, 103, 109, 110, 273, 278, 279

BPS Brier-Peyrin-Stern. IV, V, 32, 38, 40, 43, 44, 47, 49, 50, 74, 76, 90, 94–99, 168, 188, 246, 270, 271, 323

CAVP Cryptographic Algorithm Validation Program. 278

CBC Cipher-block Chaining. 10, 24, 26–28, 31, 42, 47, 90, 258, 270, 279

CFB Cipher Feedback. 24, 279

CMAC Cipher-based MAC. 304, 309

CMC CBC-Mask-CBC. 31

CRHF Collision-Resistant Hash Function. 29

CSP Critical Security Parameter. 284

CTR Counter Mode. 10, 24, 28, 103, 279

DES Data Encryption Standard. III, 13, 15, 17, 19, 20, 44, 74, 90, 259, 278

DH Diffie-Hellman. 258, 278, 279

DHE DH Ephemeral. 279

DRBG Deterministic Random Bit Generator. V, 40, 54–56, 77, 79, 103, 107–113, 168, 188, 246, 256, 273, 282–285, 288, 290–292, 323

DSA Digital Signature Algorithm. 258, 278, 279

ECB Electronic Codebook. 10, 24–26, 31, 258, 270

ECC Elliptic Curve Cryptosystem. 278, 279

ECDH Elliptic Curve DH. 279

ECDSA Elliptic Curve DSA. 279

ECIES Elliptic Curve Integrated Encryption Scheme. 279

ECMQV Elliptic Curve Menezes-Qu-Vanstone. 279

EME ECB-Mask-ECB. 31

FEAL Fast Data Encipherment Algorithm. 15

FFX Format-preserving Feistel-based Encryption. IV, V, 32, 38, 40–42, 71, 74, 90, 91, 168, 188, 246, 273, 322

FPE Format Preserving Encryption. 30, 31, 50

HMAC Keyed-Hashed Message Authentication Code. 304, 309

IDEA International Data Encryption Algorithm. 15

KDF Key Derivation Function. 304, 305, 307, 309

MAC Message Authentication Code. 27, 29, 42, 90, 258, 259, 270, 304, 309

MD4 Message Digest-4. 30, 278

MD5 Message Digest-5. 29, 278

NRBG Non-deterministic Random Bit Generator. 40, 282

OAEF Optimal Asymmetric Encryption Padding. 279

OCB Offset Codebook. 279

OFB Output Feedback. 24, 279

OMAC One-key MAC. 27

OWHF One-Way Hash Function. 29

PRF Pseudorandom Function. 304–306, 309

PRNG Pseudorandom Number Generator. 65, 294–296

RBG Random Bit Generator. IX, 282, 302–304

RNG Random Number Generator. 294–296

RSA Ron Rivest, Adi Shamir, Leonard Adleman. 259, 278, 279

RSAES RSA Encryption Scheme. 279

SAFER Secure And Fast Encryption Routine. 15

SHA Secure Hash Algorithm. III, 29, 30, 44, 109, 111–113, 136, 273, 279

TDES Triple DES. 44, 278

TES Tweakable Encyphering Scheme. 31

TRNG True Random Number Generator. 51

Computacionales

API Application Program Interface. VI, 60, 181, 188

ASCII American Standard Code for Information Interchange. 30, 89, 125

BSD Berkeley Software Distribution. 320

BSL Boost Software License. 87

CRUD Create, Read, Update and Delete. 168, 181

DAO Data Access Object. 77

DBLP Digital Bibliography and Library Project. 335

DRY Don't Repeat Yourself. 183

GCC GNU Compilers Collection. 86

GNU GNU is Not Unix. 86, 87, 260

GPL GNU General Public Licence. 86–88, 320

HTML Hypertext Markup Language. 88, 183, 232, 233, 236, 320

HTTP Hypertext Transfer Protocol. 149–152

JSON JavaScript Object Notation. 240

LPPL The L^AT_EX Project Public License. 332, 334, 335

PAL Perl Artistic License. 335

PDF Portable Document Format. 88, 333

PDS Public Domain Software. 334

PGF Progressive Graphics File. 333

PHP PHP: Hypertext Processor. 86

RFC Request for Comments. 83, 125, 197, 199

SDK Software Development Kit. 5

SDL Security Development Lifecycle. 58

SSL Secure Sockets Layer. 29, 167

TLS Transport Layer Security. 167

UML Unified Modeling Language. 69, 334

URL Uniform Resource Locator. 142, 145, 198, 200, 240

VCS Version Control System. 87, 88

WYSIWYG What You See Is What You Get. 332

Bancarios

BIN Bank Identifier Number. 52

CDV Card Data Vault. 50, 58, 64, 65, 71, 74, 77

CISP Cardholder Information Security Program. 2

DSS Data Security Standard. 2, 3, 5, 38, 58, 60

INN Issuer Identification Number. 32, 33, 103

MDES MasterCard Digital Enablement Service. 6

MII Major Industry Identifier. 32, 33, 273

PA Payment Application. 60

PAN Personal Account Number. 3, 5, 6, 30, 32, 38, 40, 50–52, 58–60, 62–70, 77, 101, 103, 114, 165

PCI Payment Card Industry. IV, 2–5, 10, 33, 38–40, 50, 58, 60, 63, 66–68, 247, 252, 270, 273, 278

VTS Visa Token Services. 6

De instituciones y agrupaciones

ACM Association for Computing Machinery. 331

APA American Psychological Association. 335

FIPS Federal Information Processing Standard. 17, 59, 63, 66, 165, 302

IEC International Electrotechnical Commission. 61

IEEE Institute of Electrical and Electronic Engineers. 335

ISO International Organization for Standardization. 61, 63, 66

NIST National Institute of Standards and Technology. IV, 10, 20, 30, 32, 33, 40, 41, 43, 54, 61, 63, 65, 66, 87, 103, 109, 278, 294, 296, 302, 312, 315

NSA National Security Agency. 30

NVLAP National Voluntary Laboratory Accreditation Program. 291

RAE Real Academia Española. 10

SSC Security Standard Council. IV, 4, 5, 10, 33, 38–40, 63, 66, 68, 247, 252, 270, 273, 278

Bibliografía

«*Those who believe in telekinetics, raise
my hand.*»

KURT VONNEGUT.

- [1] John S. Kiernan. *Credit Card And Debit Card Fraud Statistics*. [HTTPS://WALLETHUB.COM/EDU/CREDIT-DEBIT-CARD-FRAUD-STATISTICS/25725/](https://www.wallethub.com/edu/credit-debit-card-fraud-statistics/25725/). Consultado en marzo de 2018 (vid. pág. 2).
- [2] UK Cards Association. *What is PCI DSS?* [HTTP://WWW.THEUKCARDSASSOCIATION.ORG.UK/SECURITY/WHAT_IS_PCI%20DSS.ASP](http://www.theukcardsassociation.org.uk/security/WHAT_IS_PCI%20DSS.ASP). Consultado en febrero de 2018 (vid. pág. 2).
- [3] SearchSecurity Staff. *The history of the PCI DSS standard: A visual timeline*. [HTTPS://SEARCHSECURITY.TECHTARGET.COM/FEATURE/THE-HISTORY-OF-THE-PCI-DSS-STANDARD-A-VISUAL-TIMELINE](https://searchsecurity.techtarget.com/feature/The-history-of-the-PCI-DSS-standard-A-visual-timeline). Consultado en marzo de 2018 (vid. pág. 2).
- [4] David Khan. *The Codebreakers: A Comprehensive History of Secret Communication from Ancient Times to the Internet*. Nueva York: Scribner, 1996 (vid. pág. 3).
- [5] Simon Singh. *The Code Book: How to Make It, Break It, Hack It, Crack it*. Nueva York: Delacorte Press, 2001. ISBN: 0-375-89012-2 (vid. pág. 3).
- [6] Shift4 Payments. *The History of TrueTokenization*. [HTTPS://WWW.SHIFT4.COM/DOTN/4TIFY/TRUETOKENIZATION.CFM](https://www.shift4.com/dotn/4tify/TRUETOKENIZATION.CFM). Consultado en agosto de 2018 (vid. pág. 5).
- [7] Shift4 Payments. *Tried and True Tokenization: The Original Tokenization Solution for Card Data Security*. [HTTPS://WWW.SHIFT4.COM/PDF/SHIFT4-TRUETOKENIZATION.PDF](https://www.shift4.com/PDF/SHIFT4-TRUETOKENIZATION.PDF). Consultado en marzo de 2018 (vid. pág. 5).
- [8] BluePay Processing, LLC. *What you need to know about tokenization technology and BluePay's TokenShield*. [HTTPS://WWW.BLUEPAY.COM/INFOGRAPHIC/TOKENIZATION/](https://www.bluepay.com/infographic/tokenization/). Consultado en marzo de 2018 (vid. pág. 5).
- [9] Braintree. *Tokenization Secures CC Data and Meet PCI Compliance Requirements*. [HTTPS://WWW.BRAINTREEPAYMENTS.COM/BLOG/USING-TOKENIZATION-TO-SECURE-CREDIT-CARD-DATA-AND-MEET-PCI-COMPLIANCE-REQUIREMENTS/](https://www.braintreepayments.com/blog/using-tokenization-to-secure-credit-card-data-and-meet-pci-compliance-requirements/). Consultado en marzo de 2018 (vid. pág. 5).
- [10] MerchantLink. *Transaction Vault Tokenization*. [HTTP://WWW.MERCHANTLINK.COM/SOLUTIONS/PRODUCTS/TRANSACTIONVAULT/](http://www.merchantlink.com/solutions/products/TRANSACTIONVAULT/). Consultado en marzo de 2018 (vid. pág. 6).
- [11] Jack Henry & Associates Inc. *Card Processing Solutions For Jack Henry Banks*. [HTTPS://WWW.JACKHENRY.COM/JHA-PAYMENT-SOLUTIONS/CPS-JACK-HENRY-BANKS/PAGES/TOKENIZATION.ASPX](https://www.jackhenry.com/jha-payment-solutions/cps-jack-henry-banks/pages/tokenization.aspx). Consultado en marzo de 2018 (vid. pág. 6).
- [12] MasterCard. *Digital Commerce Solutions*. [HTTPS://WWW.MASTERCARD.US/EN-US/ISSUERS/PRODUCTS-AND-SOLUTIONS/GROW-MANAGE-YOUR-BUSINESS/DIGITAL-COMMERCE-SOLUTIONS.HTML](https://www.mastercard.us/en-us/issuers/products-and-solutions/grow-manage-your-business/digital-commerce-solutions.html). Consultado en marzo de 2018 (vid. pág. 6).
- [13] MasterCard. *Frequently Asked Questions*. [HTTPS://WWW.MASTERCARD.US/EN-US/FREQUENTLY-ASKED-QUESTIONS.HTML#TOKENIZATION](https://www.mastercard.us/en-us/frequently-asked-questions.html#TOKENIZATION). Consultado en marzo de 2018 (vid. pág. 6).
- [14] Securosis. *Understanding and Selecting a Tokenization Solution*. [HTTPS://SECUROSIS.COM/ASSETS/LIBRARY/REPORTS/SECUROSIS_UNDERSTANDING_TOKENIZATION_V.1_.0_.PDF](https://securosis.com/assets/library/reports/securosis_understanding_tokenization_v.1_.0_.pdf). Consultado en febrero de 2018 (vid. pág. 6).

- [15] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. 2009. URL: [HTTPS://BITCOIN.ORG/BITCOIN.PDF](https://bitcoin.org/bitcoin.pdf) (vid. págs. 7, 252).
- [16] Nitin Hardeniya, Jacob Perkins, Deepti Chopra y col. *Natural Language Processing: Python and NLTK*. Packt, 2006 (vid. pág. 7).
- [17] Andrew W. Appel. *Modern Compiler Implementation in C*. Cambridge University Press, 2004 (vid. pág. 7).
- [18] Rosabeth Moss Kanter. *Men and Women of the Corporation*. Basic Books, 1993 (vid. pág. 7).
- [19] Alfred Menezes, Paul C. van Oorschot y Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996. ISBN: 0-8493-8523-7 (vid. págs. 10, 14, 24, 28, 250, 255).
- [20] Hans Delfs y Helmut Knebl. *Introduction to Cryptography - Principles and Applications*. Information Security and Cryptography. Springer, 2007. ISBN: 978-3-540-49243-6. DOI: 10.1007/3-540-49244-5. URL: [HTTPS://DOI.ORG/10.1007/3-540-49244-5](https://doi.org/10.1007/3-540-49244-5) (vid. págs. 10, 14, 28).
- [21] Claude E. Shannon. «Communication theory - Exposition of fundamentals». En: *Trans. of the IRE Professional Group on Information Theory (TIT)* 1 (1953), págs. 44-47. DOI: 10.1109/TIT.1953.1188568. URL: [HTTPS://DOI.ORG/10.1109/TIT.1953.1188568](https://doi.org/10.1109/TIT.1953.1188568) (vid. pág. 11).
- [22] Phillip Rogaway. *A Synopsis of Format-Preserving Encryption*. 2010. URL: [HTTP://WEB.CS.UCDAVIS.EDU/~ROGAWAY/PAPERS/SYNOPSIS.PDF](http://web.cs.ucdavis.edu/~rogaway/papers/synopsis.pdf) (vid. págs. 17, 31).
- [23] Bruce Schneier y John Kelsey. «Unbalanced Feistel Networks and Block Cipher Design». En: *Fast Software Encryption, Third International Workshop, Cambridge, UK, February 21-23, 1996, Proceedings*. Ed. por Dieter Gollmann. Vol. 1039. Lecture Notes in Computer Science. Springer, 1996, págs. 121-144. ISBN: 3-540-60865-6. DOI: 10.1007/3-540-60865-6_49. URL: [HTTPS://DOI.ORG/10.1007/3-540-60865-6_49](https://doi.org/10.1007/3-540-60865-6_49) (vid. pág. 17).
- [24] Ross J. Anderson y Eli Biham. «Two Practical and Provably Secure Block Ciphers: BEARS and LION». En: *Fast Software Encryption, Third International Workshop, Cambridge, UK, February 21-23, 1996, Proceedings*. Ed. por Dieter Gollmann. Vol. 1039. Lecture Notes in Computer Science. Springer, 1996, págs. 113-120. ISBN: 3-540-60865-6. DOI: 10.1007/3-540-60865-6_48. URL: [HTTPS://DOI.ORG/10.1007/3-540-60865-6_48](https://doi.org/10.1007/3-540-60865-6_48) (vid. pág. 17).
- [25] Stefan Lucks. «Faster Luby-Rackoff Ciphers». En: *Fast Software Encryption, Third International Workshop, Cambridge, UK, February 21-23, 1996, Proceedings*. Ed. por Dieter Gollmann. Vol. 1039. Lecture Notes in Computer Science. Springer, 1996, págs. 189-203. ISBN: 3-540-60865-6. DOI: 10.1007/3-540-60865-6_53. URL: [HTTPS://DOI.ORG/10.1007/3-540-60865-6_53](https://doi.org/10.1007/3-540-60865-6_53) (vid. pág. 17).
- [26] Debrup Chakraborty y Francisco Rodríguez-Henríquez. «Block Cipher Modes of Operation from a Hardware Implementation Perspective». En: *Cryptographic Engineering*. Ed. por Çetin Kaya Koç. Springer, 2009, págs. 321-363. ISBN: 978-0-387-71816-3. DOI: 10.1007/978-0-387-71817-0_12. URL: [HTTPS://DOI.ORG/10.1007/978-0-387-71817-0_12](https://doi.org/10.1007/978-0-387-71817-0_12) (vid. pág. 24).

- [27] Alaa Hussein Al-Hamami y Ghossoon M. Waleed al-Saadoo. *Handbook of research on threat detection and countermeasures in network security*. 1.^a ed. IGI Global, 2015 (vid. pág. 28).
- [28] Prakash C. Gupta. *Cryptography and Network Security*. PHI Learning, 2015 (vid. pág. 28).
- [29] Morris Dworkin. *NIST Special Publication 800-38G - Recommendation for Block Cipher Modes of Operation: Methods for Format-Preserving Encryption*. 2016. URL: <HTTP://DX.DOI.ORG/10.6028/NIST.SP.800-38G> (vid. págs. 32, 41, 43).
- [30] Donald E. Knuth. *The Art of Computer Programming, Volume II: Seminumerical Algorithms*. Addison-Wesley, 1969. ISBN: 0201038021. URL: <HTTP://WWW.WORLDCAT.ORG/OCLC/310551264> (vid. pág. 32).
- [31] International Organization for Standardization. *ISO/IEC 7812*. 5.^a ed. 2017, pág. 7. URL: <HTTPS://WWW.ISO.ORG/STANDARD/70484.HTML> (vid. pág. 32).
- [32] International Organization for Standardization. *ISO 9362*. 4.^a ed. 2014, pág. 6. URL: <HTTPS://WWW.ISO.ORG/STANDARD/60390.HTML> (vid. pág. 32).
- [33] Abhay Bhargav. *PCI compliance: The Definitive Guide*. CRC Press, 2015 (vid. pág. 32).
- [34] Elaine Barker y John Kelsey. *NIST Special Publication 800-90A - Recommendation for Random Number Generation Using Deterministic Random Bit Generators*. 2015. URL: <HTTP://DX.DOI.ORG/10.6028/NIST.SP.800-90AR1> (vid. págs. 33, 54, 63, 65, 88, 278, 283).
- [35] Andrew Rukhin, Juan Soto, James Nechvatal y col. *NIST Special Publication 800-22 Revision 1a - A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*. 2010. URL: <HTTPS://NVLPUBS.NIST.GOV/NISTPUBS/LEGACY/SP/NISTSPECIALPUBLICATION800-22R1A.PDF> (vid. págs. 34, 109, 295, 296).
- [36] Lily Chen. *NIST Special Publication 800-108 - Recommendation for Key Derivation Using Pseudorandom Functions*. 2009. URL: <HTTP://NVLPUBS.NIST.GOV/NISTPUBS/LEGACY/SP/NISTSPECIAL-PUBLICATION800-108.PDF> (vid. págs. 34, 302).
- [37] Elaine Barker y Allen Roginsky. *NIST Special Publication 800-133 - Recommendation for Cryptographic Key Generation*. 2012. URL: <HTTP://DX.DOI.ORG/10.6028/NIST.SP.800-133> (vid. págs. 34, 302).
- [38] Elaine Barker. *NIST Special Publication 800-57 - Recommendation for Key Management*. 2016. URL: <HTTP://DX.DOI.ORG/10.6028/NIST.SP.800-57PT1R4> (vid. págs. 34, 61, 312, 315).
- [39] Elaine Barker, Miles Smid, Dennis Branstad y col. *NIST Special Publication 800-130 - A Framework for Designing Cryptographic Key Management Systems*. 2013. URL: <HTTP://DX.DOI.ORG/10.6028/NIST.SP.800-130> (vid. págs. 34, 61).
- [40] Payment Card Industry Security Standards Council. *Tokenization Product Security Guidelines – Irreversible and Reversible Tokens*. 2015. URL: HTTPS://WWW.PCISECURITYSTANDARDS.ORG/DOCUMENTS/TOKENIZATION_PRODUCT_SECURITY_GUIDELINES.PDF (vid. págs. 38, 58, 63, 65, 66, 278).

- [41] Payment Card Industry Security Standards Council. *Data Security Standard - Version 3.2*. 2016. URL: [HTTPS://WWW.PCISECURITYSTANDARDS.ORG/DOCUMENTS/PCI_DSS_V3-2.PDF](https://www.pcisecuritystandards.org/documents/PCI_DSS_v3-2.pdf) (vid. págs. 38, 60).
- [42] Mihir Bellare, Phillip Rogaway y Terence Spies. «The FFX Mode of Operation for Format-Preserving Encryption». Ver. 1.0. En: (2009). Presentado al NIST para estandarización (vid. pág. 41).
- [43] Mihir Bellare, Phillip Rogaway y Terence Spies. «The FFX Mode of Operation for Format-Preserving Encryption». Ver. 1.1. En: (2010). Prueba (vid. pág. 42).
- [44] Eric Brier, Thomas Peyrin y Jacques Stern. «BPS: a Format-Preserving Encryption Proposal». En: (2010). Presentado al NIST para estandarización (vid. pág. 43).
- [45] Sandra Diaz-Santiago, Lil María Rodríguez-Henríquez y Debrup Chakraborty. «A cryptographic study of tokenization systems». En: *Int. J. Inf. Sec.* 15.4 (2016), págs. 413-432. DOI: 10.1007/s10207-015-0313-x. URL: [HTTPS://DOI.ORG/10.1007/S10207-015-0313-X](https://doi.org/10.1007/s10207-015-0313-x) (vid. págs. 50, 51, 77).
- [46] Riccardo Aragona, Riccardo Longo y Massimiliano Sala. «Several proofs of security for a tokenization algorithm». En: *Appl. Algebra Eng. Commun. Comput.* 28.5 (2017), págs. 425-436. DOI: 10.1007/s00200-017-0313-3. URL: [HTTPS://DOI.ORG/10.1007/S00200-017-0313-3](https://doi.org/10.1007/s00200-017-0313-3) (vid. pág. 52).
- [47] Information Technology Laboratory National Institute of Standards y Technology. *FIPS PUB 140-2 - Security Requirements for cryptographic modules*. 2001. URL: [HTTPS://CSRC.NIST.GOV/CSRC/MEDIA/PUBLICATIONS/FIPS/140/2/FINAL/DOCUMENTS/FIPS1402.PDF](https://csrc.nist.gov/csrc/media/publications/fips/140/2/final/documents/fips1402.pdf) (vid. págs. 59, 63, 66, 302).
- [48] Payment Card Industry Security Standards Council. *Payment Application Data Security Standard - Requirements and Security Assessment Procedures - Version 3.0*. 2013. URL: [HTTPS://WWW.PCISECURITYSTANDARDS.ORG/MINISITE/EN/DOCS/PA-DSS_V3.PDF](https://www.pcisecuritystandards.org/minisite/en/docs/PA-DSS_v3.pdf) (vid. págs. 60, 61).
- [49] Simon Josefsson. «The Base16, Base32, and Base64 Data Encodings». En: *RFC 4648* (2006), págs. 1-18. DOI: 10.17487/RFC4648. URL: [HTTPS://DOI.ORG/10.17487/RFC4648](https://doi.org/10.17487/RFC4648) (vid. págs. 83, 335).
- [50] Gael Hofemeier y Robert Chesebrough. *Introduction to Intel AES-NI and Intel Secure Key Instructions*. [HTTPS://SOFTWARE.INTEL.COM/SITES/DEFAULT/FILES/M/D/4/1/D/8/INTRODUCTION_TO_INTEL_SECURE_KEY_INSTRUCTIONS.PDF](https://software.intel.com/sites/default/files/m/d/4/1/d/8/introduction_to_intel_secure_key_instructions.pdf). Consultado en abril de 2018. 2014 (vid. págs. 88, 89).
- [51] Mozilla Foundation. *Regular Expressions*. [HTTPS://DEVELOPER.MOZILLA.ORG/EN-US/DOCS/WEB/JAVASCRIPT/GUIDE/REGULAR_EXPRESSIONS](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions). Consultado en agosto de 2018 (vid. pág. 124).
- [52] Peter W. Resnick. «Internet Message Format». En: *RFC 5322* (2008), págs. 1-57. DOI: 10.17487/RFC5322. URL: [HTTPS://DOI.ORG/10.17487/RFC5322](https://doi.org/10.17487/RFC5322) (vid. págs. 125, 197, 199).
- [53] Django Software Foundation. *Models*. [HTTPS://DOCS.DJANGOPROJECT.COM/EN/2.1/TOPICS/DB/MODELS/](https://docs.djangoproject.com/en/2.1/topics/db/models/). Consultado en octubre de 2018 (vid. pág. 181).
- [54] Kevin D. Smith, Jim J. Jewett, Skip Montanaro y col. *PEP 318 – Decorators for Functions and Methods*. [HTTPS://WWW.PYTHON.ORG/DEV/PEPS/PEP-0318/](https://www.python.org/dev/peps/pep-0318/). Consultado en octubre de 2018 (vid. pág. 181).

- [55] Erich Gamma, Richard Helm, Ralph Johnson y col. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994. ISBN: 0-201-63361-2 (vid. pág. 183).
- [56] Martin Fowler. *Inversion of Control Containers and the Dependency Injection pattern*. [HTTPS://MARTINFOWLER.COM/ARTICLES/INJECTION.HTML](https://MARTINFOWLER.COM/ARTICLES/INJECTION.HTML). Consultado en octubre de 2018 (vid. pág. 183).
- [57] Google Inc. *Injector Documentation*. [HTTPS://DOCS.ANGULARJS.ORG/API/AUTO/SERVICE/\\$INJECTOR](https://DOCS.ANGULARJS.ORG/API/AUTO/SERVICE/$INJECTOR). Consultado en octubre de 2018 (vid. pág. 183).
- [58] Django Software Foundation. *Design philosophies*. [HTTPS://DOCS.DJANGOPROJECT.COM/EN/2.1/MISC/DESIGN-PHILOSOPHIES/](https://DOCS.DJANGOPROJECT.COM/EN/2.1/MISC/DESIGN-PHILOSOPHIES/). Consultado en octubre de 2018 (vid. pág. 183).
- [59] Google Inc. *Understanding Components*. [HTTPS://DOCS.ANGULARJS.ORG/GUIDE/COMPONENT](https://DOCS.ANGULARJS.ORG/GUIDE/COMPONENT). Consultado en octubre de 2018 (vid. pág. 236).
- [60] William Stallings. *Cryptography and network security - principles and practice* (6. ed.) Pearson, 2014. ISBN: 978-0-13-335469-0 (vid. pág. 250).
- [61] Grady Booch, Robert A. Maksimchuk, Michael W. Engle y col. *Object-oriented analysis and design with applications, Third Edition*. Addison Wesley object technology series. Addison-Wesley, 2007. ISBN: 978-0-201-89551-3 (vid. pág. 250).
- [62] Donald Beaver, Silvio Micali y Phillip Rogaway. «The Round Complexity of Secure Protocols (Extended Abstract)». En: *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*. Ed. por Harriet Ortiz. ACM, 1990, págs. 503-513. ISBN: 0-89791-361-2. DOI: 10.1145/100216.100287. URL: <HTTP://DOI.ACM.ORG/10.1145/100216.100287> (vid. págs. 251, 253).
- [63] Payment Card Industry Security Standards Council. *Data Security Standard (DSS) and Payment Application Data Security Standard (PA-DSS) - Glossary of Terms, Abbreviations, and Acronyms - Version 1.2*. 2008. URL: HTTPS://WWW.PCISECURITYSTANDARDS.ORG/PDFS/PCI_DSS_GLOSSARY.PDF (vid. pág. 252).
- [64] Sanjay Bhattacherjee y Palash Sarkar. «Cryptocurrency Voting Games». En: *IACR Cryptology ePrint Archive* 2017 (2017), pág. 1167. URL: <HTTP://EPRINT.IACR.ORG/2017/1167> (vid. pág. 252).
- [65] Alfred V. Aho y Jeffrey D. Ullman. *Foundations of Computer Science, C Edition*. Computer Science Press / W. H. Freeman, 1992. ISBN: 0716782332. URL: <HTTP://I.STANFORD.EDU/%5C%7EULLMAN/FOCS.HTML> (vid. págs. 253, 335).
- [66] Marvin Minsky. *Computation: finite and infinite machines*. Prentice-Hal, 1967 (vid. pág. 254).
- [67] Melanie Mitchell. *Complexity - A Guided Tour*. Oxford University Press, 2009. ISBN: 978-0-19-512441-5. URL: <HTTP://UKCATALOGUE.OUP.COM/PRODUCT/9780195124415.D0> (vid. pág. 254).
- [68] Martin D. Davis. *Computability and Unsolvability*. McGraw-Hill Series in Information Processing and Computers. McGraw-Hill, 1958 (vid. pág. 254).

- [69] Ian Sommerville. *Software engineering, 8th Edition*. International computer science series. Addison-Wesley, 2007. ISBN: 9780321313799. URL: [HTTP://WWW.WORLDCAT.ORG/OCLC/65978675](http://www.worldcat.org/oclc/65978675) (vid. pág. 255).
- [70] Tobias Oetiker, Hubert Partl, Irene Hyna y col. *The Not So Short Introduction to LaTeX*. [HTTPS://TOBI.OETIKER.CH/LSHORT/LSHORT.PDF](https://tobi.oetiker.ch/lshort/lshort.pdf). Consultado en octubre de 2018 (vid. pág. 332).
- [71] Dieter Gollmann, ed. *Fast Software Encryption, Third International Workshop, Cambridge, UK, February 21-23, 1996, Proceedings*. Vol. 1039. Lecture Notes in Computer Science. Springer, 1996. ISBN: 3-540-60865-6. DOI: [10.1007/3-540-60865-6](https://doi.org/10.1007/3-540-60865-6). URL: [HTTPS://DOI.ORG/10.1007/3-540-60865-6](https://doi.org/10.1007/3-540-60865-6).

Lista de figuras

2.1. CLASIFICACIÓN DE LA CRIPTOGRAFÍA	12
2.2. CANAL DE COMUNICACIÓN CON CRIPTOGRAFÍA SIMÉTRICA.	12
2.3. CANAL DE COMUNICACIÓN CON CRIPTOGRAFÍA ASIMÉTRICA.	13
2.4. DIAGRAMA GENÉRICO DE UNA RED FEISTEL.	16
2.5. GENERALIZACIONES DE LAS REDES FEISTEL.	18
2.6. DIAGRAMA DE LA OPERACIÓN <i>SubBytes</i>	21
2.7. DIAGRAMA DE LA OPERACIÓN <i>ShiftRows</i>	22
2.8. DIAGRAMA DE LA OPERACIÓN <i>MixColumns</i>	22
2.9. DIAGRAMA DE LA OPERACIÓN <i>AddRoundKey</i>	23
2.10. MODO DE OPERACIÓN ECB.	25
2.11. MODO DE OPERACIÓN CBC.	26
2.12. CBC-MAC.	27
2.13. DIAGRAMA DEL FUNCIONAMIENTO DE UNA FUNCIÓN HASH.	28
2.14. COMPONENTES DE UN NÚMERO DE TARJETA.	33
3.1. CLASIFICACIÓN DE LOS TOKENS SEGÚN PAYMENT CARD INDUSTRY (PCI) SECURITY STANDARD COUNCIL (SSC).	39
3.2. CLASIFICACIÓN PROPUESTA DE LOS TOKENS.	39
3.3. CORRIEMIENTO DE CURSOR PARA LA SELECCIÓN DEL ÚLTIMO BLOQUE EN EL MODO DE OPERACIÓN DE BRIER-PEYRIN-STERN (BPS).	47
3.4. MODO DE OPERACIÓN DE BPS.	49
4.1. DIAGRAMA DE DEPENDENCIAS ENTRE PAQUETES.	71
4.2. DIAGRAMA DE CLASES GENERAL.	72

4.3. DIAGRAMA DE COMPONENTES DE PROGRAMA.	73
4.4. DIAGRAMA DE CLASES DE MÓDULO DE FFX.	75
4.5. DIAGRAMA DE CLASES DE MÓDULO DE BPS.	76
4.6. DIAGRAMA DE CLASES DE MÓDULO DE TKR.	78
4.7. DIAGRAMA DE CLASES DE MÓDULO DE AHR.	80
4.8. DIAGRAMA DE CLASES DE MÓDULO DE DRBG.	81
4.9. DIAGRAMA DE CLASES DE LA ESTRUCTURA DE PRUEBAS.	82
4.10. CLASE DE CODIFICADOR.	83
5.1. COMPARACIÓN DE TIEMPOS ENTRE ALGORITMOS TOKENIZADORES.	116
5.2. COMPARACIÓN DE TIEMPOS ENTRE ALGORITMOS TOKENIZADORES REVERSIBLES.	117
5.3. COMPARACIÓN DE TIEMPOS ENTRE ALGORITMOS TOKENIZADORES IRREVERSIBLES.	118
5.4. COMPARACIÓN DE TIEMPOS DE TOKENIZACIÓN Y DETOKENIZACIÓN.	119
6.1. DIAGRAMA DE ESTADOS DE ACTORES.	126
6.2. DIAGRAMA DE ESTADOS DE UN CLIENTE.	127
6.3. DIAGRAMA DE ESTADOS DE TOKENS.	128
6.4. DIAGRAMA DE ESTADOS DE LLAVES.	129
6.5. DIAGRAMA DE ESTADOS DE UN CORREO ELECTRÓNICO.	130
6.6. DIAGRAMA GENERAL DE CASOS DE USO.	139
6.18. MODELO DE DATOS DE APLICACIÓN.	166
6.19. DIAGRAMA ENTIDAD-RELACIÓN DE LA BASE DE DATOS DE LA APLICACIÓN.	169
6.20. DIAGRAMA RELACIONAL DE LA BASE DE DATOS.	170
6.21. DIAGRAMA DE CLASES DE APLICACIÓN WEB.	171
6.22. DIAGRAMA DE SECUENCIA PARA REGISTRAR A UN CLIENTE.	172

6.23. DIAGRAMA DE SECUENCIA PARA OPERACIÓN DE TOKENIZACIÓN	173
6.24. DIAGRAMA DE SECUENCIA PARA OPERACIONES GENÉRICAS.	174
7.1. COMPARACIÓN DE TIEMPOS DE OPERACIÓN.	189
8.1. DIAGRAMA GENERAL DE CASOS DE USO DE TIENDA DE LIBROS EN LÍNEA.	195
8.15. DIAGRAMA ENTIDAD-RELACIÓN PARA LA TIENDA EN LÍNEA.	223
8.16. DIAGRAMA RELACIONAL DE LA BASE DE DATOS.	224
8.17. DIAGRAMA DE CLASES LA TIENDA EN LÍNEA.	225
8.18. DIAGRAMA DE SECUENCIA PARA REGISTRAR A UN CLIENTE.	226
8.19. DIAGRAMA DE SECUENCIA PARA AGREGAR UN LIBRO AL CARRITO.	227
8.20. DIAGRAMA DE SECUENCIA PARA REALIZAR UNA COMPRA.	228
8.21. DIAGRAMA DE ESTADOS PARA UN CLIENTE.	229
8.22. DIAGRAMA DE ESTADOS PARA UNA TARJETA.	229
8.23. DIAGRAMA DE ESTADOS PARA UNA DIRECCIÓN.	229
D.1. DIAGRAMA DEL <i>counter mode</i>	305
D.2. DIAGRAMA DEL <i>feedback mode</i>	307
D.3. DIAGRAMA DEL <i>double pipeline mode</i>	308
E.1. DIAGRAMA DE ESTADO DE LLAVES CRIPTOGRÁFICAS.	317
F.1. COMPARACIÓN DE ALGORITMOS REVERSIBLES	324
F.2. COMPARACIÓN DE ALGORITMOS IRREVERSIBLES	325

Lista de tablas

1. NOTACIÓN	XII
2.1. IDENTIFICADOR DE INDUSTRIA (MAJOR INDUSTRY IDENTIFIER (MII))	33
3.1. COLECCIÓN DE PARÁMETROS FORMAT-PRESERVING FEISTEL-BASED ENCRYPTION (FFX) A10	42
4.1. RESUMEN DE REQUERIMIENTOS DEL PAYMENT CARD INDUSTRY (PCI) SECURITY STANDARD COUNCIL (SSC) PARA LOS SISTEMAS TOKENIZADORES	68
5.1. RESULTADO DE LAS PRUEBAS ESTADÍSTICAS DEL DETERMINISTIC RANDOM BIT GENERATOR (DRBG) BASADO EN CIFRADOS POR BLOQUE (ADVANCED ENCRYPTION STANDARD (AES)) PARA LOS NIVELES DE SEGURIDAD DE 112 Y 128.	110
5.2. RESULTADO DE LAS PRUEBAS ESTADÍSTICAS DEL DRBG BASADO EN CIFRADOS POR BLOQUE (AES) PARA LOS NIVELES DE SEGURIDAD DE 192 Y 256.	110
5.3. RESULTADO DE LAS PRUEBAS ESTADÍSTICAS DEL DRBG BASADO EN FUNCIONES HASH (SECURE HASH ALGORITHM (SHA)256) PARA LOS NIVELES DE SEGURIDAD DE 112 Y 128.	111
5.4. RESULTADO DE LAS PRUEBAS ESTADÍSTICAS DEL DRBG BASADO EN FUNCIONES HASH (SHA256) PARA LOS NIVELES DE SEGURIDAD DE 192 Y 256.	112
5.5. RESULTADO DE LAS PRUEBAS ESTADÍSTICAS DEL DRBG BASADO EN FUNCIONES HASH (SHA512) PARA LOS NIVELES DE SEGURIDAD DE 112 Y 128.	113
5.6. RESULTADO DE LAS PRUEBAS ESTADÍSTICAS DEL DRBG BASADO EN FUNCIONES HASH (SHA512) PARA LOS NIVELES DE SEGURIDAD DE 192 Y 256.	113
5.7. COMPARACIÓN DE TIEMPOS DE TOKENIZACIÓN.	114
6.1. RELACIÓN DE CASOS DE USO Y REQUERIMIENTOS	138
7.1. COMPARACIÓN DE TIEMPOS.	188
A.1. LONGITUDES DE LLAVE MÍNIMAS Y MODOS DE OPERACIÓN PERMITIDOS PARA ALGORITMOS CRIPTOGRÁFICOS	279

A.2. ALGORITMOS HASH PERMITIDOS	279
E.1. CLASIFICACIÓN DE LLAVES CRIPTOGRÁFICAS	313
E.2. CRIPTOPERIODOS SUGERIDOS POR TIPO DE LLAVE	315

Apéndices

« “Begin at the beginning,” the King said,
very gravely, “and go on till you came to
the end: then stop.”»

LEWIS CARROLL.

Apéndice A

Requerimientos para las primitivas criptográficas

«Lock up your libraries if you like; but there is no gate, no lock, no bolt that you can set upon the freedom of my mind.»

Virginia Woolf, A Room of One's Own.

En este apéndice se resumen los requerimientos mínimos que debe de tener cualquier PRIMITIVA CRIPTOGRÁFICA que se use dentro del sistema *tokenizador*. Esta información se presenta en el anexo C de [40], el cual está clasificado como *informativo* solamente (no *normativo*). Con respecto a las PRIMITIVAS CRIPTOGRÁFICAS, la parte *normativa* está controlada por el programa CRYPTOGRAPHIC ALGORITHM VALIDATION PROGRAM (CAVP) del NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY (NIST); el cual evalúa las implementaciones usadas de una serie de primitivas comunes¹.

En la tabla A.1 se colocan los tamaños mínimos de llaves y MODOS DE OPERACIÓN (sección 2.2.6) asociados para las PRIMITIVAS CRIPTOGRÁFICAS de los «algoritmos criptográficos»² permitidos. Estos son: ADVANCED ENCRYPTION STANDARD (AES) (sección 2.2.5), RON RIVEST, ADI SHAMIR, LEONARD ADLEMAN (RSA) (sección 2.1.2), ELLIPTIC CURVE CRYPTOSYSTEM (ECC) y DIGITAL SIGNATURE ALGORITHM (DSA)/DIFFIE-HELLMAN (DH). Se hace especial énfasis en que TRIPLE DES (TDES) no está permitido.

La tabla A.2 enlista los algoritmos hash (sección 2.3) permitidos. Para evitar introducir fallas de seguridad a través de las funciones hash, estas deben proveer al menos tantos bits de seguridad como el algoritmo criptográfico usado, y en cualquier caso, no menos de 128 bits (lo que deja fuera a MESSAGE DIGEST-4 (MD4) y MESSAGE DIGEST-5 (MD5)).

El número de bits de entropía utilizados para los generadores de números aleatorios debe de ser mayor o igual al número de bits de seguridad utilizados para las primitivas anteriores. Cuando se utilicen generadores determinísticos, estos deben seguir las recomendaciones del NIST en [34].

¹Esta lista se puede encontrar en [HTTPS://CSC.NIST.GOV/PROJECTS/CRYPTOGRAPHIC-ALGORITHM-VALIDATION-PROGRAM](https://csrc.nist.gov/projects/crypto-validation-program)

²El PAYMENT CARD INDUSTRY (PCI) SECURITY STANDARD COUNCIL (SSC) parece dividir a las PRIMITIVAS CRIPTOGRÁFICAS en *algoritmos criptográficos, funciones hash y generadores de números pseudoaleatorios*; esto resulta confuso dado que las tres categorías pertenecen al campo de estudio de la criptografía.

Algoritmo	Tamaño de llave	Modo de operación
AES	128	CTR, OCB, CBC, OFB, CFB
RSA	3072	RSAES-OAEP
ECC	256	ECDH, ECMQV, ECDSA, ECIES
DSA/DH	3072/256	DHE

Tabla A.1: Longitudes de llave mínimas y MODOS DE OPERACIÓN permitidos para algoritmos criptográficos

Bits de seguridad	Algoritmo hash
128	SECURE HASH ALGORITHM (SHA)-256
128	SHA3-256
192	SHA3-384
256	SHA-512
256	SHA3-512

Tabla A.2: Algoritmos hash permitidos

Apéndice B

Generación de bits pseudoaleatorios

«*Heme aquí, ya al final, y todavía no sé
qué cara le daré a la muerte.*»

Rosario Castellanos.

Existen dos maneras de generar bits aleatorios: la primera es producir bits de manera no determinística, donde el estado de cada uno (uno o cero) está determinado por un proceso físico impredecible. Estos generadores de bits aleatorios (RANDOM BIT GENERATOR (RBG)) son conocidos como generadores no determinísticos, o NON-DETERMINISTIC RANDOM BIT GENERATOR (NRBG). La otra manera, que será explorada a continuación, es calcular determinísticamente los bits mediante un algoritmo; estos generadores determinísticos son conocidos como DETERMINISTIC RANDOM BIT GENERATOR (DRBG).

Un DRBG tiene un mecanismo que utiliza un algoritmo que produce una secuencia de bits partiendo de un valor inicial que es determinado por una SEMILLA que, a su vez, está determinada por la salida de la fuente de aleatoriedad. Una vez que se tiene la SEMILLA y se determina el valor inicial, el DRBG es instanciado y puede producir valores. Dado a su naturaleza determinística, se dice que los valores producidos por el DRBG son pseudoaleatorios y no aleatorios; si la SEMILLA es mantenida oculta y el algoritmo fue bien diseñado, los bits de salida del DRBG serán impredecibles.

La entrada de ENTROPÍA es provista a un mecanismo DRBG para obtener una SEMILLA utilizando una fuente de aleatoriedad. La entrada de ENTROPÍA y la SEMILLA deben mantenerse secretas; que estos valores permanezcan secretos es una de las bases de la seguridad del DRBG. Otras entradas, como un NONCE o una cadena de personalización pueden ser utilizadas como entradas; estas pueden o no requerir ser mantenidas secretas también, y ser utilizadas para crear la SEMILLA inicial para el DRBG.

El estado interno es la memoria del DRBG y consiste en todos los valores que requiere el mecanismo (parámetros, variables, etcétera).

El mecanismo DRBG requiere cinco funciones; estas son explicadas con más detalle abajo:

1. Instanciación (*stantiate function*): obtiene la entrada de ENTROPÍA para crear una SEMILLA con la cual será creado un nuevo estado interno. La entrada puede ser combinada con una NONCE o una cadena de personalización.
2. Generación (*generate function*): genera bits pseudoaleatorios utilizando el estado interno actual; también tiene como salida un nuevo estado interno que es utilizado para el siguiente pedido.
3. Cambio de SEMILLA (*reseed function*): obtiene una nueva entrada de ENTROPÍA y la combina con el estado interno actual para crear una nueva SEMILLA y un nuevo estado interno.
4. Desinstanciación (*unstantiate function*): elimina el estado interno actual.
5. Prueba de salud (*health test function*): determina que el mecanismo DRBG siga funcionando correctamente.

Cuando a un DRBG se le aplica la función de cambio de SEMILLA, es imperativo que la SEMILLA sea distinta a la que se utilizó en la función de instanciaión. Cada SEMILLA define un nuevo periodo de

SEMILLA (*seed period*) para la instanciación del DRBG. Una instanciación consiste en uno o más períodos de SEMILLA, estos comienzan cuando se obtiene una nueva SEMILLA y terminan cuando la siguiente SEMILLA es obtenida o el DRBG deja de utilizarse.

El estado interno deriva de la SEMILLA; este incluye el estado de trabajo (uno o más valores derivados de la SEMILLA que deben permanecer secretos, y la cuenta con el número de salidas que se han producido con esa semilla) y la información administrativa (el nivel de seguridad, etc). Es menester proteger el estado interno del DRBG. La implementación del mecanismo DRBG puede haber sido diseñado para tener múltiples instancias; en este caso, cada instancia debe tener su propio estado interno y el estado interno de una instancia DRBG jamás debe ser utilizado como estado interno para una instancia distinta. El estado interno no debe ser accesible a funciones distintas a las cinco del DRBG, ni a otras instancias del DRBG o a otros DRBG.

Los mecanismos especificados en [34] soportan cuatro niveles de seguridad: 112, 128, 192 y 256 bits. Este es uno de los parámetros que se necesitan para instanciar un DRBG; además, dependiendo de su diseño, cada mecanismo DRBG tiene sus restricciones de nivel de seguridad. El nivel de seguridad depende de la implementación del DRBG y la cantidad de ENTROPÍA que se da como entrada a la función de instanciación.

Los bits pseudoaleatorios obtenidos mediante un DRBG no deben ser utilizados por una aplicación que requiera un nivel mayor de seguridad que con el que fue instanciado el DRBG. La concatenación de dos salidas del DRBG tampoco proveen un nivel de seguridad más alto que del que fueron instanciados (por ejemplo, dos cadenas concatenadas de 128 bits no dan como resultado una cadena de 256 bits con el nivel de seguridad de 256 bits).

B.1. Semillas

Las SEMILLAS deben ser obtenidas antes de generar bits pseudoaleatorios en el DETERMINISTIC RANDOM BIT GENERATOR (DRBG), pues esta es utilizada para instanciar al DRBG y determinar el estado inicial interno del mecanismo.

Cambiar la SEMILLA restaura el secreto de la salida del DRBG si el estado interno o la SEMILLA son conocidos. Hacer este cambio periódicamente es una buena manera de mantener a raya el peligro de que valores como la entrada de ENTROPÍA, la SEMILLA o el estado interno de trabajo; hayan sido comprometidos.

Los ingredientes para determinar una nueva SEMILLA para la función de instanciación son la entrada de entropía de una fuente aleatoria, un NONCE y una cadena de personalización (recomendada, pero no obligatoria). Para hacer un cambio de semilla, se necesita el estado interno actual, la entrada de ENTROPÍA y una entrada adicional opcional.

La longitud de la SEMILLA depende del mecanismo DRBG y el nivel de seguridad requerido; sin

embargo, siempre debe ser de, al menos, el mismo número de bits de ENTROPÍA requerida.

La entrada de ENTROPÍA y la SEMILLA resultante deben de ser protegidas con el mismo cuidado con el que se protege la salida del DRBG; por ejemplo, si el mecanismo es utilizado para generar llaves, estos valores deben protegerse con la misma seguridad como son protegidas las llaves generadas. Además, la seguridad del DRBG depende de mantener en secreto la entrada de ENTROPÍA, por lo que esa entrada debe ser tratada como un parámetro crítico de seguridad (CRITICAL SECURITY PARAMETER (CSP)) y ser obtenido desde un módulo criptográfico que contenga la función necesaria o ser transmitido desde un canal seguro.

Cuando se requiera un NONCE para la construcción de la SEMILLA, esta debe cumplir con una de las siguientes dos condiciones: tener *nivel_seguridad*/2 bits de ENTROPÍA o un valor que se espera no se repita más de lo que se repetiría una cadena aleatoria de *nivel_seguridad*/2 bits. Aunque no debe ser mantenido en secreto, cada NONCE debe ser considerado como un CSP y debe ser único en el módulo criptográfico en donde se realiza la instancia. El NONCE puede estar compuesto por uno o más de los siguientes valores:

1. Valor aleatorio generado para cada NONCE por un generador de bits aleatorios aprobado.
2. Una marca de tiempo con la resolución suficiente para que sea distinto cada vez que sea utilizado.
3. Un número de secuencia que se incremente constantemente.
4. Una combinación de una marca de tiempo y un número de secuencia que se incremente constantemente; tal que el número de secuencia regrese a su valor inicial solo cuando la marca de tiempo cambie.

Generar demasiadas salidas partiendo de una misma SEMILLA puede proveer suficiente información para ser capaz de predecir las salidas futuras; por lo que el cambio de SEMILLAS reduce riesgos de seguridad. Las SEMILLAS tienen una vida finita que depende el mecanismo DRBG utilizado. Es imperativo que las implementaciones respeten el límite de la vida de las SEMILLAS especificado para el mecanismo; y, cuando se alcance el límite de la vida de una SEMILLA, el DRBG no debe generar salidas hasta que se haya cambiado la SEMILLA o se cree una nueva instancia del DRBG (aunque se prefiere que se cambie la SEMILLA). Una SEMILLA jamás debe ser utilizada para inicializar o cambiar la semilla de otra instancia del DRBG o la suya.

La cadena de personalización (que es opcional pero recomendada) es utilizada para derivar la SEMILLA, puede ser obtenida dentro o fuera del módulo criptográfico y hasta puede ser una cadena vacía, pues el DRBG no depende de esta cadena para obtener ENTROPÍA. De hecho, que el adversario conozca la cadena de personalización no disminuye el nivel de seguridad de una instancia de DRBG siempre y cuando la entrada de ENTROPÍA se mantenga desconocida. Esta cadena no es considerada un CSP; puede introducir datos adicionales al DRBG, tales como identificadores de usuario, aplicación, versiones, protocolos, marcas de tiempo, direcciones de red, números de serie, etcétera.

B.2. Funciones

Un DETERMINISTIC RANDOM BIT GENERATOR (DRBG) necesita cinco funciones para poder funcionar correctamente.

B.2.1. Instanciación

Antes de generar bits pseudoaleatorios, el DETERMINISTIC RANDOM BIT GENERATOR (DRBG) debe ser instanciado; esta función se encarga de revisar que los parámetros de entrada sean válidos, determina el nivel de seguridad para la instancia de DRBG que se generará, obtiene la entrada de ENTROPÍA capaz de soportar el nivel de seguridad y el NONCE (solo si es requerido), determina el estado interno inicial y, si se tienen varias instancias del DRBG simultáneas, obtiene un manejador de estado. Las entradas de la función son las siguientes:

1. Nivel de seguridad requerido.
2. Bandera que indica si se necesita o no la resistencia de predicción.
3. Cadena de personalización.
4. Entrada de ENTROPÍA.
5. NONCE

Las primeras entradas deben ser provistas por la aplicación consumidora. La salida de la función de instanciación a esta aplicación consiste en:

1. El estado del proceso; regresa un *EXITOSO* o un estado inválido indicando el error que hubo al instanciar al DRBG.
2. El manejador de estado, utilizado para identificar el estado interno de esta instancia para generar, cambiar la SEMILLA y demás funciones.

El algoritmo de la función de instanciación se puede observar en el pseudocódigo B.1.

B.2.2. Cambio de SEMILLA

Cambiar la SEMILLA de una instancia no es requerido, pero es recomendado que se realice este proceso cada que sea posible. Cambiar la SEMILLA puede:

- Ser solicitado expresamente por la aplicación consumidora.

```

1  entrada: nivel_seguridad_requerido, bandera_prediccion, cadena_personalizacion
2          entrada_entropia, nonce
3  salida: estado, manejador_estado
4  inicio
5      si nivel_seguridad_requerido > mayor_nivel_seguridad_soportado:
6          regresar BANDERA_ERROR,invalido
7      si bandera_prediccion =verdadero Y resistencia_prediccion no es soportado:
8          regresar BANDERA_ERROR,invalido
9      si longitud(cadena_personalizacion) > longitud_maxima:
10         regresar BANDERA_ERROR,invalido
11     Asignar a nivel_seguridad el nivel más bajo de seguridad mayor o igual
12         a nivel_seguridad_requerido del conjunto {112,128,192,256}
13     (estado,entrada_entropia) = obtener_entropia(nivel_seguridad,...
14         ... longitud_min,longitud_max,bandera_prediccion)
15     si (estado ≠ EXITOSO):
16         regresar estado,invalido
17     Obtener nonce
18     estado_trabajo_inicial = INSTANCIAR_ALGORITMO(entrada_entropia,...
19         ... nonce,cadena_personalizacion,nivel_seguridad)
20     Obtener manejador_estado para el estado interno vacío.
21     si no se encuentra un estado interno vacío:
22         regresar BANDERA_ERROR,invalido
23     Configurar el estado interno de la nueva instancia con los valores iniciales
24         y la información administrativa.
25     regresar (EXITOSO,manejador_estado)
26 fin

```

Pseudocódigo B.1: DRBG, instanciación.

- Realizado cuando la aplicación consumidora requiere resistencia de predicción.
- Disparada por la función generadora cuando se alcanza un número predeterminado de salidas generadas.
- Disparada por eventos externos.

La función se encarga de revisar la validez de los parámetros de entrada, obtiene la entrada de ENTROPÍA de una fuente de aleatoriedad y, mediante el algoritmo de cambio de SEMILLA, combina el estado de trabajo actual con la nueva entrada de ENTROPÍA y valores adicionales para determinar el nuevo estado de trabajo actual. Las entradas para la función de cambio de SEMILLA son las siguientes:

1. El manejador de estado.
2. Bandera que indica si se requiere o no la resistencia de predicción.
3. Entradas adicionales (opcionales).
4. Entrada de ENTROPÍA.
5. Valores del estado interno e información administrativa.

Las primeras tres entradas deben ser provistas por la aplicación consumidora. La salida consiste en lo siguiente:

1. Estado que regresa la función.
2. Nuevo estado de trabajo interno.

El proceso para cambiar la SEMILLA se observa en el pseudocódigo B.2.

B.2.3. Generación

Esta función es utilizada para generar bits pseudoaleatorios después de haber utilizado la función de instanciación o de cambio de SEMILLA. Se encarga de validar los parámetros de entrada, llamar a la función de cambio de SEMILLA cuando sea requerida ENTROPÍA extra, generar los bits pseudoaleatorios, actualizar el estado de trabajo y regresar los bits a la aplicación consumidora que los pidió. La función tiene las siguientes entradas:

1. Manejador de estado.
2. El número de bits que se requieren.
3. El nivel de seguridad que se necesita.

```

1 entrada: bandera_prediccion , manejador_estado , entrada_adicional
2           entrada_entropia , estado_interno
3 salida: estado , estado_trabajo_interno
4 inicio
5   si manejador_estado indica un estado inválido o sin uso:
6     regresar BANDERA_ERROR
7   si se requiere resistencia de predicción y bandera_prediccion =falso:
8     regresar BANDERA_ERROR
9   si longitud(entrada_adicional) > longitud_max_entrada_adicional:
10     regresar BANDERA_ERROR
11   (estado,entrada_entropia) = obtener_entropia(nivel_seguridad, ...
12       ... longitud_min,longitud_max,bandera_prediccion)
13   si (estado ≠ EXITOSO):
14     regresar estado
15   estado_trabajo_nuevo = ALGORITMO_CAMBIAR_SEMILLA(estado_trabajo, ...
16       ... entrada_entropia,entrada_adicional)
17   Reemplazar el valor de estado_trabajo con estado_trabajo_nuevo
18   regresar (EXITOSO)
19 fin

```

Pseudocódigo B.2: DRBG, cambio de semilla.

4. Si debe ser resistente a predicción o no.
5. Entradas adicionales.
6. El estado de trabajo actual e información administrativa.

Después de haber sido generado, la salida consiste en lo siguiente

1. Estado.
2. Bits pseudoaleatorios.
3. Nuevo estado de trabajo.

El proceso para generar bits se puede observar en el pseudocódigo B.3.

Hay que tomar en cuenta cuando la implementación no tiene la capacidad de hacer el cambio de SEMILLA en el algoritmo: se quitan los pasos 6 y 7, y, si el estado indica que se necesita hacer el cambio, se regresa un error que indique que el DETERMINISTIC RANDOM BIT GENERATOR (DRBG) no puede seguir siendo utilizado y hay que eliminar la instancia. También se debe tener en mente cuando se termina el ciclo de vida de la SEMILLA: cada que se llama al algoritmo generador, se revisa si el contador ha alcanzado

```

1  entrada: bandera_prediccion , manejador_estado , entrada_adicional
2          nivel_seguridad_requerido , no_bits_requeridos , estado_interno
3  salida: estado , estado_trabajo , bits_pseudoaleatorios
4  inicio
5      si manejador_estado indica un estado inválido o sin uso:
6          regresar (BANDERA_ERROR,NULL)
7      si no_bits > no_bits_maximo
8          regresar (BANDERA_ERROR,NULL)
9      si nivel_seguridad_requerido > nivel_seguridad indicado en el estado interno:
10         regresar (BANDERA_ERROR,NULL)
11      si longitud(entrada_adicional) > longitud_max_entrada_adicional:
12          regresar (BANDERA_ERROR,NULL)
13      si se requiere resistencia de predicción y bandera_prediccion =falso:
14          regresar (BANDERA_ERROR,NULL)
15  Limpiar bandera_cambio_sevilla_necesario
16  si bandera_cambio_sevilla_necesario o bandera_prediccion están puestas:
17      estado = CAMBIO_SEMILLA(manejador_estado,prediccion_resistencia,entrada_adicional)
18      si estado ≠ EXITOSO:
19          regresar (estado,NULL)
20  Obtener el nuevo estado interno
21  entrada_adicional = NULL
22  Poner en cero bandera_cambio_sevilla_necesario
23  (estado,bits_pseudoaleatorios,nuevo_estado_trabajo) = ALGORITMO_GENERAR(estado_trabajo, ...
24  ... no_bits_requeridos,entrada_adicional)
25  si estado indica que se requiere cambiar la semilla antes de poder generar bits:
26      Activar bandera_cambio_sevilla_necesario
27      si se requiere resistencia de predicción:
28          Activar bandera_prediccion
29          Regresar al paso 7.
30  Reemplazar el estado_trabajo con nuevo_estado_trabajo.
31  regresar (EXITOSO,bits_pseudoaleatorios)
32  fin

```

Pseudocódigo B.3: DRBG, generación.

```
1 entrada: manejador_estado
2 salida: estado
3 inicio
4   si manejador_estado indica un estado inválido:
5     regresar (BANDERA_ERROR)
6     Eliminar los contenidos del estado interno indicados por manejador_estado
7     regresar (EXITOSO)
8 fin
```

Pseudocódigo B.4: DRBG, desinstanciación.

el valor máximo que se encuentra en el estado interno; en caso de ser así, la función debe avisar que se requiere un cambio de SEMILLA.

B.2.4. Desinstanciación

Esta función se encarga de liberar el estado interno de una instancia al borrar el contenido de su estado interno. La función requiere del manejador de estado de la instancia que se va a eliminar y regresa el estado de la función. El proceso para eliminar una instancia se puede observar en el pseudocódigo B.4.

B.3. Mecanismos basados en funciones hash

Los mecanismos DETERMINISTIC RANDOM BIT GENERATOR (DRBG) pueden estar basados en funciones hash de un solo sentido; dos mecanismos basados en estas funciones son los HASH_DRBG y HMAC_DRBG. El nivel de seguridad que puede soportar cada uno es el nivel de resistencia a la PREIMAGEN que tiene la función hash (véase sección 2.3).

El mecanismo HASH_DRBG requiere el uso de una misma función hash en las funciones de instanciación, cambio de SEMILLA y generación. El nivel de seguridad de la función hash a utilizar debe igualar o ser mayor al nivel que se requiere por la aplicación consumidora del DRBG. En cambio, el mecanismo HMAC_DRBG utiliza múltiples ocurrencias de una función hash con llave; la misma función hash debe ser utilizada a lo largo del proceso de instanciación. Al igual que HASH_DRBG, la función hash debe tener, al menos, el mismo nivel de seguridad que la aplicación consumidora requiere para la salida del DRBG.

B.4. Mecanismos basados en cifrados por bloque

Un cifrado por bloques DETERMINISTIC RANDOM BIT GENERATOR (DRBG) está basado en un algoritmo de cifrado por bloques. La seguridad que puede alcanzar el DRBG depende del cifrado por bloques y el tamaño de llave utilizado. Se tiene al mecanismo CTR_DRBG, que utiliza un cifrado por bloques aprobado con el modo de operación de contador (véase sección 2.2.6); debe utilizarse el mismo algoritmo de cifrado y tamaño de llave para todas las operaciones de cifrado por bloques en el DRBG. El CTR_DRBG tiene una función actualizadora que es llamada por los algoritmos de instanciación, generación y cambio de semilla para ajustar el estado interno cuando hay nueva ENTROPÍA, se le dan entradas adicionales o cuando se actualiza el estado interno después de generar bits pseudoaleatorios.

B.5. Garantías

Los usuarios de un DETERMINISTIC RANDOM BIT GENERATOR (DRBG) requieren una garantía de que el generador en verdad produce bits pseudoaleatorios, que el diseño, implementación y uso de servicios criptográficos son adecuados para proteger la información del usuario y, finalmente, necesita una garantía de que el generador sigue operando correctamente. La implementación debe ser validada por un laboratorio acreditado por NATIONAL VOLUNTARY LABORATORY ACCREDITATION PROGRAM (NVLAP) para tener la certeza de que el mecanismo está bien implementado. Se requiere, además, para garantizar el funcionamiento del mecanismo, lo siguiente:

Documentación mínima se debe proveer, al menos, el siguiente conjunto de documentos; una gran parte podrían pertenecer al manual de usuario:

1. Documentación del método para obtener la entrada de entropía.
2. Documentar cómo la implementación fue diseñada para permitir la validación de la implementación y revisar su estado.
3. Documentar el tipo de mecanismo DRBG y las primitivas criptográficas utilizadas.
4. Documentar los niveles de seguridad soportados por la implementación.
5. Documentar las características que soporta la implementación.
6. Si las funciones del DRBG están distribuidas, especificar los mecanismos que se usan para proteger la confidencialidad e integridad de las partes del estado interno que son trasnferidas entre las partes distribuidas del DRBG.
7. Indicar si se utiliza una función de derivación; si no se utiliza, documentar que la implementacion solo puede utilizarse cuando la entrada de ENTROPÍA completa está disponible.
8. Documentar todas las funciones de soporte.
9. Si se requieren hacer pruebas periódicas para la función generadora, documentar los intervalos y justificarlos.

10. Documentar si las funciones del DRBG pueden ser puestas a prueba sobre demanda.

Pruebas de validación de la implementación El mecanismo DRBG debe ser diseñado para ser probado y poder asegurar que el producto está correctamente implementado. Debe proveerse una interfaz para realizarle pruebas que permita insertar los datos de entrada y obtener los datos de salida. Todas las funciones que se incluyen en la implementación deben ser probadas en la funcionalidad de las pruebas de salud.

Pruebas de salud La implementación DRBG debe realizarse pruebas de salud a sí mismo para asegurarse de que continua operando correctamente. Se deben realizar pruebas del tipo *known answer*, donde se tiene una entrada para la que ya se sabe la respuesta correcta.

Manejo de errores Se indica para cada función del mecanismo qué errores son los esperados; es menester indicar el tipo de error que ocurrió.

Apéndice C

Pruebas estadísticas para generadores de números aleatorios y pseudoaleatorios

«*Someone has to protect this family from
the man who protect this family.*»

Breaking Bad.

En este apéndice se explican los puntos más importantes del estándar del NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY (NIST) referente a las pruebas estadísticas para los generadores de números aleatorios, NIST *Special Publication 800-22 Revision 1a - A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*, descrito en **nist:pruebas**.

C.1. Definiciones

Existen dos tipos básicos de generadores, los RANDOM NUMBER GENERATOR (RNG) y los PSEUDORANDOM NUMBER GENERATOR (PRNG).

C.1.1. Generadores aleatorios

Este tipo de generador usa fuentes de ENTROPÍA no determinísticas para producir números aleatorios, dichas fuentes de ENTROPÍA normalmente consisten en la medición de eventos físicos, como el ruido en un circuito eléctrico.

La salidas de los RANDOM NUMBER GENERATOR (RNG) puede usarse directamente para obtener números aleatorios, o como una entrada de un PSEUDORANDOM NUMBER GENERATOR (PRNG). En el primer caso, es necesario que la salida satisfaga pruebas estadísticas para determinar que dicha salida parece aleatoria.

Los RNG que se usan con fines criptográficos tienen que ser impredecibles (ver sección C.1.4), por lo cual es recomendado combinar varias fuentes de ENTROPÍA, dado que algunas (como los vectores de tiempo) son demasiado predecibles. De cualquier forma, en la mayoría de los casos es mejor usar los PRNG para obtener de manera directa números aleatorios.

C.1.2. Generadores pseudoaleatorios

Estos son generadores de múltiples números aleatorios a partir de una o varias entradas llamadas SEMILLA. Cuando es necesario que la salida sea impredecible, la SEMILLA debe ser aleatoria y también impredecible por sí sola, razón por lo cual esta se debe de obtener por medio de un RANDOM NUMBER GENERATOR (RNG).

La salida de los PSEUDORANDOM NUMBER GENERATOR (PRNG) normalmente son funciones determinísticas de la SEMILLA, por lo que la verdadera aleatoriedad queda en la generación de esta misma, motivo por el que se usa el término de *pseudoaleatorio* y lo que causa la necesidad de mantener en secreto la SEMILLA.

Las mejores cualidades de los PRNG son que, al contrario de lo que se creería por el nombre, suelen parecer más aleatorias que las RNG, así como que producen números aleatorios más rápido.

C.1.3. Aleatoriedad

Una secuencia de bits aleatoria puede interpretarse como una secuencia en la que el valor de cada bit es independiente, y los valores que lo conforman están uniformemente distribuidos (ver DISTRIBUCIÓN UNIFORME).

C.1.4. Impredicibilidad

Los generadores aleatorios y pseudoaleatorios deben de ser impredecibles. La impredecibilidad es la incapacidad de conocer la salida de un bit $n + 1$ aun conociendo los valores del bit 0 al n si la SEMILLA es desconocida. Esta también es la incapacidad de determinar la SEMILLA usando el conocimiento de los valores generados por la misma.

Hay que resaltar que conociendo tanto la SEMILLA como el algoritmo generador, el PSEUDORANDOM NUMBER GENERATOR (PRNG) es completamente predecible, por lo cual en la mayoría de los casos el algoritmo es público y la SEMILLA se mantiene secreta.

C.1.5. Pruebas estadísticas

Estas sirven para determinar si la secuencia a evaluar puede ser considerada una secuencia aleatoria. Existe una gran diversidad de pruebas estadísticas para comparar y evaluar una secuencia generada por un RANDOM NUMBER GENERATOR (RNG) o un PSEUDORANDOM NUMBER GENERATOR (PRNG) contra una secuencia verdaderamente aleatoria, cada una de estas buscando la existencia o ausencia de patrones para poder indicar si la secuencia parece o no aleatoria.

Es necesario aclarar que ningún conjunto de pruebas puede ser considerado completo, y que el resultado de determinada prueba es complementado por otros, motivo por el cual los resultados obtenidos deben ser interpretados cuidadosamente para no llegar a conclusiones incorrectas.

En [35] se determina que las pruebas estadísticas están formuladas para determinar si una secuencia binaria cumple con la hipótesis nula (H_0), o la hipótesis alternativa (H_a), las cuales dicen que la secuencia probada es aleatoria y no aleatoria respectivamente. Cada prueba descrita en la sección C.2 consiste en aceptar alguna de las dos hipótesis mencionadas.

Para cada prueba, es necesario elegir una medida estadística relevante de aleatoriedad, para así establecer una DISTRIBUCIÓN DE PROBABILIDAD y un valor crítico que sirvan de referencia, y durante la prueba, se pueda comparar el valor estadístico de la secuencia probada con el valor elegido; aceptando la hipótesis nula en caso de que no se exceda del valor crítico y rechazándola en caso contrario. De manera práctica, las pruebas estadísticas funcionan dado que la DISTRIBUCIÓN DE PROBABILIDAD de referencia y el valor crítico son seleccionados asumiendo que la secuencia es aleatoria.

Hay que mencionar que estas pruebas estadísticas pueden tener dos tipos de errores, que la secuencia sea aleatoria y se concluya que no lo es, (*error tipo I*), y que la secuencia no sea aleatoria y se concluya que sí, (*error tipo II*). La probabilidad de que ocurra el error de tipo I es normalmente llamada *nivel de significancia* de la prueba, y se denota como α , este valor dentro de la criptografía es común que ronde cerca de 0.01. La probabilidad de que ocurra el error de tipo II se denota con β y a diferencia de α , no es un valor fijo y es más difícil de calcular.

Uno de los principales objetivos de las pruebas es minimizar la probabilidad de β , por lo cual se selecciona un tamaño de muestra n y un valor para α , para después establecer un valor crítico que produzca el valor β más pequeño. Esto es posible gracias a que las probabilidades α y β están relacionadas entre sí y con el tamaño de muestra n .

Cada prueba se basa en un valor estadístico de prueba calculado, que es una función de los mismos datos, por lo cual si el valor estadístico de prueba es S y el valor crítico es t , entonces:

$$\alpha = P(S > t \mid H_0 \text{ es verdadera}) = P(\text{rechazar } H_0 \mid H_0 \text{ es verdadero}) \quad (\text{C.1})$$

$$\beta = P(S \leq t \mid H_0 \text{ es falso}) = P(\text{aceptar } H_0 \mid H_0 \text{ es falso}) \quad (\text{C.2})$$

Las pruebas estadísticas calculan un valor P , que indica la probabilidad de que un RNG perfecto haya producido una secuencia menos aleatoria que la secuencia que se probó, dado el tipo de aleatoriedad evaluada por la prueba. En caso de que el valor de P sea de 1 en una prueba, se considera que la secuencia parece tener una aleatoriedad perfecta y en caso de que el valor sea 0, la secuencia parece ser por completo no aleatoria.

Eligiendo un nivel de significancia α para las pruebas, si $P \geq \alpha$, entonces la hipótesis nula es aceptada, es decir, la secuencia parece ser aleatoria; y si $P < \alpha$, entonces la hipótesis nula es rechazada, es decir, la secuencia parece ser no aleatoria. Normalmente el valor de α se elige en el rango de [0.001, 0.01].

C.2. Pruebas

El paquete de pruebas descrito en el estándar del NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY (NIST) 800-22R1A: *Statistical Suite for Random Number Generators*, disponible en [35], es un conjunto de 15 algoritmos que se encargan de probar la aleatoriedad de secuencias binarias de longitudes arbitrariamente largas producidas por un RANDOM NUMBER GENERATOR (RNG) o un PSEUDORANDOM NUMBER GENERATOR (PRNG).

C.2.1. Prueba de frecuencia

(*The frequency (Monobit) Test*)

Esta prueba se encarga de medir la proporción de ceros y unos en toda la secuencia binaria. Su propósito es determinar si la cantidad de ceros y de unos es parecida a la que se esperaría en una secuencia verdaderamente aleatoria. Algo a tener en cuenta es que todos las pruebas subsecuentes dependen de pasar esta prueba.

C.2.2. Prueba de frecuencia en un bloque

(*Frequency Test within a Block*)

Esta prueba se centra en medir la proporción de unos dentro de N bloques de M -bits de la secuencia a probar. El propósito de la prueba es saber si la frecuencia de unos se aproxima a $M/2$ como sería de esperar de una secuencia aleatoria.

C.2.3. Prueba de carreras

(*The Runs Test*)

Esta prueba se encarga de calcular el número de carreras en la secuencia que se está probando, donde una carrera es una subsecuencia ininterrumpida de bits del mismo valor encerrada entre bits del valor opuesto. El objetivo de la prueba es determinar si el número de carreras en la secuencia es parecido al que se esperaría de una secuencia aleatoria, particularmente, establece si la oscilación entre ceros y unos es muy rápida o muy lenta. Cabe resaltar que la prueba tiene como requisito pasar la prueba de frecuencia.

C.2.4. Prueba de la carrera más larga en un bloque

(*Tests for the Longest-Run-of-Ones in a Block*)

El propósito de esta prueba es determinar si la longitud de la carrera más larga en la secuencia es consistente con la longitud de la carrera más larga de una secuencia aleatoria. Se tiene que resaltar que encontrar una irregularidad en la carrera más larga de unos implica una irregularidad en la carrera más larga de ceros, por lo cual solo es necesario hacer la prueba para la carrera más larga de unos.

C.2.5. Prueba del rango de matriz binaria

(*The Binary Matrix Rank Test*)

La prueba tiene como finalidad revisar si existen DEPENDENCIAS LINEALES entre subcadenas de longitud fija de la secuencia a probar.

C.2.6. Prueba Espectral

(*The Discrete Fourier Transform (Spectral) Test*)

Esta prueba se encarga de obtener los picos más altos de la transformada discreta de Fourier de la secuencia. Su propósito es detectar comportamientos periódicos en la secuencia que indiquen que esta no puede asumirse como aleatoria. De manera más precisa, la prueba detecta si el número de picos que exceden un umbral del 95 % son significativamente diferentes al 5 %.

C.2.7. Prueba de coincidencia sin superposición

(*The Non-overlapping Template Matching Test*)

La prueba tiene como finalidad detectar generadores que producen una gran cantidad de ocurrencias de un patrón no periódico en sus secuencias de salida. Para esta prueba se utiliza una ventana de m bits que recorre la secuencia, para buscar patrones específicos de m bits. Dado que esta prueba es sin traslape, en caso de que se encuentre el patrón se recorre la ventana m bits y de lo contrario se recorre uno.

C.2.8. Prueba de coincidencia con superposición

(*The Overlapping Template Matching Test*)

La prueba tiene como finalidad detectar generadores que producen una gran cantidad de ocurrencias de un patrón no periódico en sus secuencias de salida. Para esta prueba se utiliza una ventana de m bits que recorre la secuencia, para buscar patrones específicos de m bits. Dado que esta prueba es con traslape, independientemente de si se encuentra o no un patrón, la ventana se recorre un bit.

C.2.9. Prueba estadística universal de Maurer

(*Maurer's "Universal Statistical" Test*)

Esta prueba se centra en medir el número de bits existentes entre patrones que coinciden, medida que se relaciona con el tamaño mínimo para comprimir una secuencia. La prueba tiene como finalidad determinar si una secuencia puede comprimirse de manera significativa sin perder información, lo cual significa que no es aleatoria.

C.2.10. Prueba de complejidad lineal

(*The Linear Complexity Test*)

El objetivo de la prueba es determinar si la secuencia a probar es lo bastante compleja como para considerarse aleatoria, esto por medio de medir el tamaño de su REGISTRO DE DESPLAZAMIENTO CON RETROALIMENTACIÓN LINEAL correspondiente, ya que las secuencias aleatorias se caracterizan por tener valores altos en este parámetro.

C.2.11. Prueba serial

(*The Serial Test*)

Esta prueba tiene como objetivo determinar si el número de ocurrencias de 2^m patrones de m bits son aproximadamente las mismas que se esperarían de una cadena aleatoria, considerando que al ser una secuencia aleatoria, es una secuencia uniforme, y cada patrón de m bits tiene la misma probabilidad de aparecer que los demás.

C.2.12. Prueba de entropía aproximada

(*The Approximate Entropy Test*)

La prueba se encarga de comparar la frecuencia de sobreponer dos bloques de longitudes consecutivas con el resultado esperado de una secuencia que es aleatoria.

C.2.13. Prueba de sumas acumulativas

(*The Cumulative Sums (Cusums) Test*)

Su objetivo es determinar si las sumas acumulativas de una secuencia parcial de la secuencia probada son muy grandes o pequeñas con relación a las sumas acumulativas de una secuencia aleatoria. Las sumas acumulativas son consideradas como caminatas aleatorias, y para las secuencias aleatorias la realización de una caminata aleatoria debe tener un valor cercano a cero.

C.2.14. Prueba de excursiones aleatorias

(*The Random Excursions Test*)

Esta prueba tiene como finalidad determinar si el número de apariciones de un estado x en un ciclo es parecido al número que se esperaría de una secuencia aleatoria. Esta prueba está conformada de 8 subpruebas, cada una para los valores de estado x de -4, -3, -2, -1, 1, 2, 3, 4.

C.2.15. Prueba variante de excursiones aleatorias

(*The Random Excursions Variant Test*)

La prueba se centra en calcular el número de veces que un estado en particular ocurre en una suma acumulativa de una caminata aleatoria. Su propósito, es detectar desviaciones del número esperado de ocurrencias de varios estados en una caminata aleatoria. Esta prueba está conformada de 18 subpruebas, cada una para los valores de estado x de -9 a 1 y 1 a 9.

Apéndice D

Generación de llaves

*«No tomamos decisiones, son las
decisiones las que nos toman a nosotros.»*

Todos los nombres, JOSÉ SARAMAGO.

En este apéndice se tratan los puntos más relevantes de dos estándares del NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY (NIST):

- NIST *Special Publication 800-133 - Recommendation for Cryptographic Key Generation*, descrito en [37].
- NIST *Special Publication 800-108 - Recommendation for Key Derivation Using Pseudorandom Functions*, descrito en [36].

D.1. Conceptos clave

D.1.1. Métodos para la generación de llaves

La generación de llaves se puede hacer por medio del uso de generadores de bits aleatorios (RANDOM BIT GENERATOR (RBG)), de la derivación de llaves a partir de otras, a partir de una contraseña; y del uso de un esquema de acuerdo entre llaves (*key agreement*); pero todas, de forma directa o indirecta, deben estar basadas en la salida de un RBG.

D.1.2. Dónde generar las llaves

La generación de llaves criptográficas debe ir de acuerdo con el FEDERAL INFORMATION PROCESSING STANDARD (FIPS) 140 (descrito en [47]), y, si es necesario que las llaves sean transferidas, se debe de hacer por medio de un canal seguro. Además, todos los valores aleatorios requeridos para la generación de las llaves deben ser generados dentro de un mismo módulo criptográfico.

D.1.3. Fuerza de la seguridad

La fuerza de la seguridad de un método es una medición de la complejidad asociada a la recuperación de información secreta o de la seguridad relativa a un algoritmo criptográfico a partir de datos conocidos.

Se dice que un método soporta la fuerza de seguridad, si la fuerza de seguridad provista por dicho método es igual o mayor a la seguridad requerida para la protección de la información.

- La fuerza de seguridad para los RANDOM BIT GENERATOR (RBG) está basada en la ENTROPÍA o aleatoriedad que provee el mismo RBG.
- La seguridad de un algoritmo criptográfico se basa en el hecho de que las llaves que usan fueron generadas por medio de procesos que las proveyeron de una ENTROPÍA mayor o igual a la necesaria para el algoritmo, así como el mismo tamaño de las llaves.

- La fuerza de seguridad de una llave depende del algoritmo que la utilizará, su tamaño, el proceso con el que fue generada, y la forma en la que es utilizada.

D.1.4. Usos de las salidas de los RANDOM BIT GENERATOR (RBG)

Suponiendo que K es una llave simétrica o un valor aleatorio que sirve como entrada de un algoritmo generador de pares de llaves asimétricas, K debe ser una cadena de bits tal que:

$$K = U \oplus V \quad (\text{D.1})$$

Donde U es una cadena de bits que se obtuvo de un RBG con el soporte de la fuerza de seguridad requerida y V es una cadena de bits de la misma longitud, que además fue generada de manera tal que es independiente de U y viceversa. La independencia entre U y V se requiere debido a que se tiene que evitar que el conocimiento de alguna de estas cadenas pueda usarse para obtener información de la otra.

D.1.5. Generación de pares de llaves asimétricas

Los pares de llaves solo pueden ser generados o por el propietario de las mismas llaves o por un tercero de confianza capaz de proveerlas.

Las llaves privadas deben permanecer secretas dentro del módulo criptográfico del propietario o de un tercero de confianza; en caso de que se tengan que transferir dichas llaves, se debe de garantizar que solo el propietario o la parte generadora puedan verlas en claro.

D.1.6. Generación de llaves simétricas

Las llaves deben de ser generadas por una o varias de las entidades que las compartirán, o por un tercero de confianza capaz de compartirlas de una manera segura.

Las llaves que son generadas directamente de un RANDOM BIT GENERATOR (RBG) deben de cumplir con la forma establecida en la ecuación D.1.

Cuando sea necesario compartir una llave, su distribución debe de ser manual, o por medio de un envolvimiento de la misma (*key wrapping*), el cual debe de soportar la fuerza de seguridad requerida para la protección de la información que protege la llave.

Obtener llaves derivadas de una contraseña es una práctica cuestionable, debido a que comúnmente la aleatoriedad en las contraseñas es mínima, por tal razón, al generarse llaves de esta manera, es fuertemente recomendado que los usuarios seleccionen contraseñas con una gran cantidad de ENTROPÍA. De cualquier

manera, se considera que este tipo de llaves proveen una ENTROPÍA nula, a menos que la contraseña haya sido generada con un RBG.

Cuando se tiene un conjunto de llaves K_1, \dots, K_n generadas de forma independiente, estas pueden ser combinadas entre sí para formar una llave K . Igualmente, si se tiene un conjunto de bloques de información V_1, \dots, V_m que son independientes de sus respectivas llaves, se pueden combinar estos bloques y sus llaves para formar otra llave.

Los métodos aprobados para generar llaves a partir de la combinación de otras son:

- La concatenación de varias llaves. $K = K_1 \parallel \dots \parallel K_n$.
- La aplicación de compuertas *xor* (o exclusiva) a varias llaves. $K = K_1 \oplus \dots \oplus K_n$.
- La aplicación de compuertas *xor* (o exclusiva) a varias llaves y bloques de información. $K = K_1 \oplus \dots \oplus K_n \oplus V_1 \oplus \dots \oplus V_m$.

Cuando sea necesario reemplazar una llave, la nueva llave debe de ser completamente independiente de la anterior, para que el conocimiento de esta última, no proporcione ningún conocimiento de la nueva.

D.2. Funciones Pseudoaleatorias (PRF)

Las funciones pseudoaleatorias o PSEUDORANDOM FUNCTION (PRF) (ver sección B) son funciones computables en TIEMPO POLINOMIAL con un índice o SEMILLA s y una variable de entrada x , de manera que cuando s se selecciona aleatoriamente de S , es COMPUTACIONALMENTE INDISTINGUIBLE de una función aleatoria definida en el mismo dominio y rango que $PRF(s, x)$.

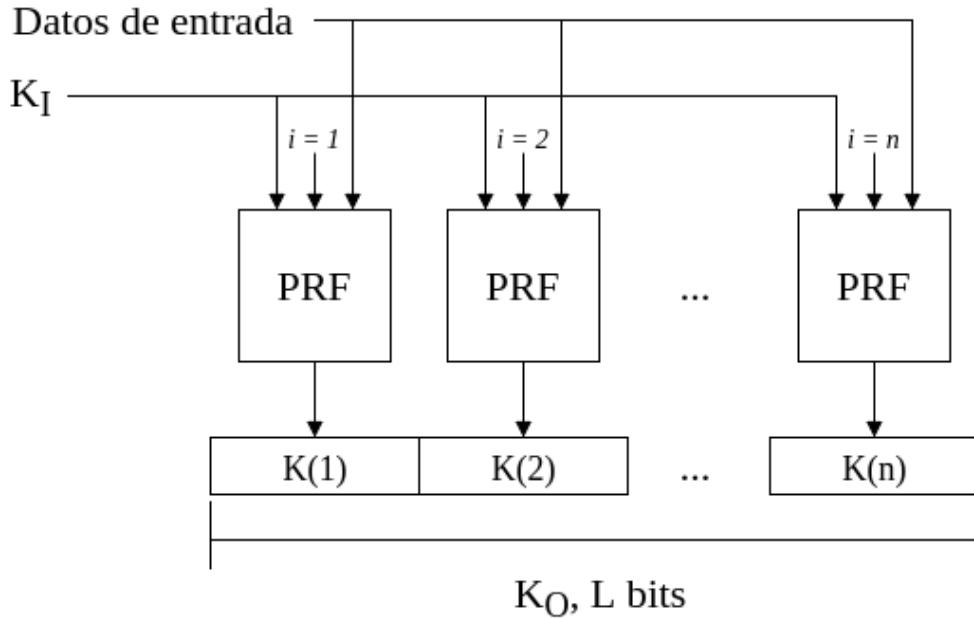
Cuando una llave criptográfica K_I es usada como la SEMILLA de una PRF, su salida se puede utilizar como MATERIAL DE LLAVES.

Para la derivación de llaves se tiene permitido el uso del KEYED-HASHED MESSAGE AUTHENTICATION CODE (HMAC) o del CIPHER-BASED MAC (CMAC) como función pseudoaleatoria.

D.3. Funciones de derivación de llaves (KDF)

Las funciones de derivación de llaves o KEY DERIVATION FUNCTION (KDF) son funciones que a partir de una llave dada como entrada, pueden generar MATERIAL DE LLAVES capaz de ser empleado por varios algoritmos criptográficos.

Las llaves de entrada de este tipo de funciones son llamadas *key derivation keys* y deben ser llaves criptográficas generadas por medio de un RANDOM BIT GENERATOR (RBG) o por un proceso automático de establecimiento de llaves.

Figura D.1: Diagrama del *counter mode*.

El MATERIAL DE LLAVES segmentado es usado como un conjunto de llaves criptográficas correspondientes a distintos algoritmos que pueden ofrecer diferentes servicios, por lo tanto las KDF deben de definir una manera de transformar este material en llaves distintas.

De forma general las KDF funcionan iterando n veces una función pseudoaleatoria para concatenar sus salidas hasta que se alcance la longitud de bits deseada para el MATERIAL DE LLAVES.

D.4. Modos de iteración

Algo necesario para el funcionamiento de las KEY DERIVATION FUNCTION (KDF) son los modos de iteración, ya que definen las entradas que se tendrán y el orden de los campos de salida en cada ciclo.

D.4.1. Counter mode

En la figura D.1 se aprecia la forma de operación de este modo de iteración, en el que los datos de entrada son concatenados en una cadena binaria de la forma: $Label \parallel 0x00 \parallel Context \parallel [L]_2$ donde el *Label* es un valor que identifica el propósito del MATERIAL DE LLAVES, $0x00$ es un octeto de ceros, el *Context* es una cadena con la información relacionada al MATERIAL DE LLAVES, y $[L]_2$ es la longitud del MATERIAL DE LLAVES que se desea obtener representada en binario. Como se observa, en este modo de iteración el contador forma parte los datos de entrada de la PSEUDORANDOM FUNCTION (PRF).

```

1 entrada: La llave  $K_I$ , la longitud  $L$  y los valores de Label y Context.
2 salida:  $K_O$ , con una longitud de  $L$  bits.
3 inicio
4     calcular  $n = \lceil \frac{L}{h} \rceil$ , donde  $h$  es la longitud de salida de la PRF.
5     si  $n > 2^r - 1$ , indicar error y parar; donde  $r \leq 32$ .
6      $resultado(0) = \emptyset$ 
7     para  $i = 1$  hasta  $n$ :
8          $K(i) = PRF(K_I, [i]_2 \parallel Label \parallel 0x00 \parallel Context \parallel [L]_2)$ .
9          $resultado(i) = resultado(i-1) \parallel K(i)$ .
10     $K_O =$  los  $L$  bits más a la izquierda del  $resultado(n)$ .
11 fin

```

Pseudocódigo D.1: Funcionamiento del *counter mode*.

```

1 entrada: La llave  $K_I$ , la longitud  $L$ , el vector de inicialización IV
2                 y los valores de Label y Context.
3 salida:  $K_O$ , con una longitud de  $L$  bits.
4 inicio
5     calcular  $n = \lceil \frac{L}{h} \rceil$ , donde  $h$  es la longitud de salida de la PRF.
6     si  $n > 2^{32} - 1$ , indicar error y parar.
7      $resultado(0) = \emptyset$ .
8      $K(0) = IV$ .
9     para  $i = 1$  hasta  $n$ :
10     $K(i) = PRF(K_I, K(i-1) \{ \parallel [i]_2 \} \parallel Label \parallel 0x00 \parallel Context \parallel [L]_2)$ .
11     $resultado(i) = resultado(i-1) \parallel K(i)$ .
12     $K_O =$  los  $L$  bits más a la izquierda del  $resultado(n)$ .
13 fin

```

Pseudocódigo D.2: Funcionamiento del *feedback mode*.

D.4.2. Feedback mode

Este modo de iteración opera tomando la salida de la PSEUDORANDOM FUNCTION (PRF) en la iteración anterior como parte los datos de entrada de la iteración actual. Cabe resaltar que como se muestra en la figura D.2, es opcional tener un contador concatenado a estos datos, que son iguales que en el modo de operación anterior, solo que con un VECTOR DE INICIALIZACIÓN *IV*.

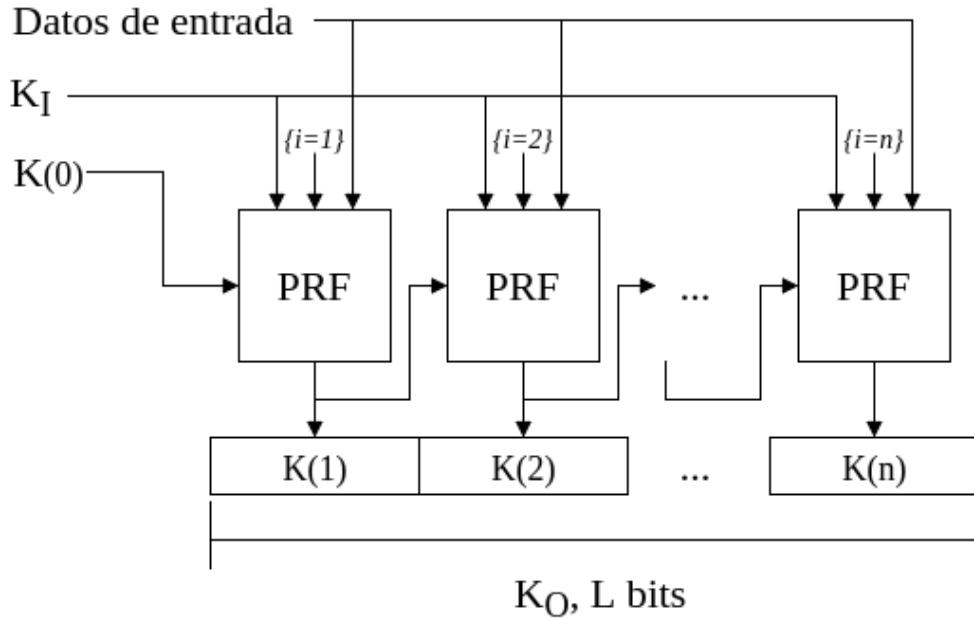


Figura D.2: Diagrama del *feedback mode*.

D.4.3. Double pipeline mode

Como su nombre lo indica y a diferencia de los otros dos modos de iteración mencionados, este usa dos flujos, donde un flujo es el encargado de generar valores secretos, y el otro usa como entradas las salidas del primero para obtener K_O . Los datos de entrada son iguales que en el primer modo de iteración mencionado, y su funcionamiento está descrito en la figura D.3.

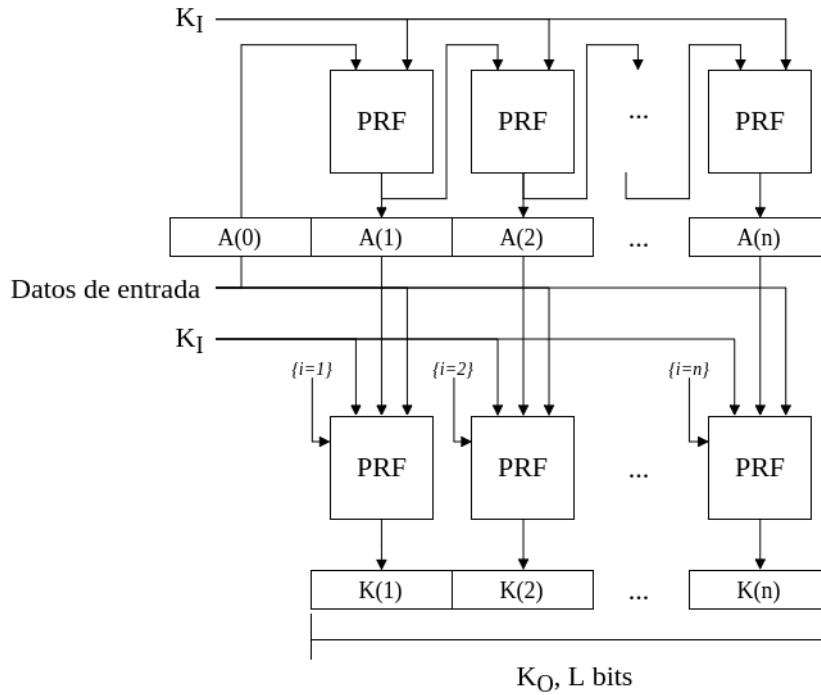
D.4.4. Jerarquía de llaves

El MATERIAL DE LLAVES proveniente de una derivación puede ser usado una o más veces como llave de entrada de otras derivaciones subsecuentes, así, es posible establecer una jerarquía en la que las llaves tienen distintos niveles.

D.5. Consideraciones de seguridad

D.5.1. Fuerza criptográfica

La fuerza de seguridad de una KEY DERIVATION FUNCTION (KDF) es medida por la cantidad de trabajo requerido para distinguir su cadena de salida de una cadena de bits que en verdad tenga una DISTRIBUCIÓN UNIFORME y cuente con la misma longitud, suponiendo que la llave de entrada K_I , es la única entrada desconocida para la KDF.


 Figura D.3: Diagrama del *double pipeline mode*.

```

1 entrada: La llave  $K_I$ , la longitud  $L$  y los valores de Label y Context.
2 salida:  $K_O$ , con una longitud de  $L$  bits.
3 inicio
4     calcular  $n = \lceil \frac{L}{h} \rceil$ , donde  $h$  es la longitud de salida de la PRF.
5     si  $n > 2^{32} - 1$ , indicar error y parar.
6      $resultado(0) = \emptyset$ .
7      $A(0) = IV = Label \parallel 0x00 \parallel Context \parallel [L]_2$ 
8     para  $i = 1$  hasta  $n$ :
9          $A(i) = PRF(K_I, A(i - 1))$ .
10         $K(i) = PRF(K_I, A(i) \{ \parallel [i]_2 \} \parallel Label \parallel 0x00 \parallel Context \parallel [L]_2)$ .
11         $resultado(i) = resultado(i - 1) \parallel K(i)$ .
12      $K_O =$  los  $L$  bits más a la izquierda del  $resultado(n)$ .
13 fin
```

Pseudocódigo D.3: Funcionamiento del *double pipeline mode*.

D.5.2. La longitud de la llave de entrada

En algunas KEY DERIVATION FUNCTION (KDF) el tamaño de la llave K_I está definido por la PSEUDO-RANDOM FUNCTION (PRF) que usan internamente, por ejemplo cuando se usa el CIPHER-BASED MAC (CMAC), pero igualmente, otras KDF pueden usar llaves de cualquier tamaño, como al usar el KEYED-HASHED MESSAGE AUTHENTICATION CODE (HMAC) como PRF.

D.5.3. Transformación de material de llaves a llaves criptográficas.

La longitud en bits del MATERIAL DE LLAVES está limitada tanto por el algoritmo relacionado a la salida de la KEY DERIVATION FUNCTION (KDF), como por el modo de iteración que se usa.

D.5.4. Codificación de los datos de entrada

La información de entrada de una KEY DERIVATION FUNCTION (KDF) consiste en diferentes campos de datos, estos deben de estar ordenados de forma específica y sin ambigüedad, de manera que se tenga un método de codificación capaz de mapear cada campo individualmente. Esto es necesario para poder detectar ataques a la KDF que dependan de la manipulación de esta información.

D.5.5. Separación entre llaves

Las llaves provenientes del MATERIAL DE LLAVES deben de estar separadas criptográficamente, de tal manera que no se comprometa la seguridad de ninguna de estas llaves derivadas. Cuando el MATERIAL DE LLAVES se obtiene de múltiples ejecuciones de la KEY DERIVATION FUNCTION (KDF) usando la misma llave de entrada K_I , la KDF debe de garantizar que el MATERIAL DE LLAVES de una ejecución no comprometa al de cualquier otra.

D.5.6. Enlace de contexto

Todo el MATERIAL DE LLAVES debe estar ligado a todas las entidades relacionadas para poder evitar errores de protocolo.

Apéndice E

Administración de llaves

«Cangrejo: ... un maravilloso concierto de piano para dos manos izquierdas. El penúltimo (y único) movimiento era una fuga a una voz. No puede imaginar lo intrincada que era. Como finale, tocamos la novena Zenfonía de Beethoven. Cuando terminó, toda la audiencia se puso de pie y aplaudió con una mano. Fue sobrecogedor.»

Gödel, Escher, Bach: An Eternal Golden
Braid, Contrafactus, DOUGLAS R.
HOFSTADTER.

En este apéndice se explican los puntos más importantes del estándar del NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY (NIST) con respecto a la administración de llaves criptográficas, NIST *Special Publication 800-57 - Recommendation for Key Management*, descrito en [38].

E.1. Tipos de llaves

Las llaves criptográficas se dividen en 19 categorías, las cuales surgen de la combinación de la naturaleza de la llave (simétrica, pública o privada) con la funcionalidad que se le da (firmas, autenticación, cifrado, entre otras). En la tabla E.1 se muestran dichas categorías: se marca con las combinaciones que tienen sentido práctico; por ejemplo, para firmar (y posteriormente validar) solamente se ocupan las llaves de un esquema asimétrico (esto es, públicas y privadas).

Para firmas: llaves que son usadas en algoritmos asimétricos para generar firmas digitales con posibles implicaciones a largo plazo. Cuando son manejadas correctamente, este tipo de llaves está diseñado para proveer AUTENTICACIÓN DE ORIGEN, autenticación de integridad y soporte para el no repudio.

Para autenticación: son usadas para proporcionar garantías sobre la identidad de un fuente generadora (esto es, AUTENTICACIÓN DE ORIGEN).

Para cifrado de datos: usadas en esquemas simétricos para proveer de confidencialidad a la información (esto es, para cifrar la información). Como se tratan de algoritmos simétricos (o de llave secreta), la misma llave se usa para quitar la confidencialidad (para descifrar).

Como envoltura de otras llaves: también llamadas llaves cifradoras de llaves (*key-encrypting keys*), son usadas para proteger otras llaves usando algoritmos simétricos. Dependiendo del algoritmo con el que se use la llave, esta también puede ser usada para proveer de validaciones de integridad.

Para generadores aleatorios: llaves usadas para generar números o bits aleatorios.

Como llaves maestra: una llave maestra simétrica se usa para derivar otras llaves simétricas (p. ej. llaves de cifrado, o de envoltura). También se conoce como llave de derivación de llaves (*key-derivation key*).

Para el transporte de llaves: llaves utilizadas en esquemas asimétricos para proteger la comunicación en un proceso de acuerdo de llaves. Se usan para proteger otras llaves (p. ej. de cifrado, de envoltura) o información relacionada (p. ej. VECTORES DE INICIALIZACIÓN).

Para el acuerdo de llaves: utilizadas en algoritmos de acuerdo de llaves para establecer otras llaves. Generalmente funcionan en plazos muy largos.

Efímeras para el acuerdo de llaves: llaves de muy corto plazo que son usadas una sola vez en un proceso de establecimiento de otras llaves. En algunos casos la llave efímera puede ser usada más de una vez, pero dentro de la misma transacción.

Funcionalidad	Naturaleza		
	Simétrica	Pública	Privada
Para firmas		✓	✓
Para autenticación	✓	✓	✓
Para cifrado de datos	✓		
Como envoltura de otras llaves	✓		
Para generadores aleatorios	✓		
Como llave maestra	✓		
Para el transporte de llaves		✓	✓
Para el acuerdo de llaves	✓	✓	✓
Efímeras para el acuerdo de llaves		✓	✓
De autorización	✓	✓	✓

Tabla E.1: Clasificación de llaves criptográficas

De autorización: usadas para proveer y verificar los privilegios de una entidad. En un esquema simétrico, la llave es conocida tanto por la entidad que provee acceso, como por la que desea acceder.

E.2. Usos de llaves

En general, las llaves se deben de utilizar para un solo propósito: el uso de la misma llave para dos operaciones criptográficas puede debilitar la seguridad provista por ambas; al limitar el uso de una llave se limita el nivel de riesgo de que esta se vea comprometida; algunos usos de llaves interfieren unos con otros.

La regla anterior no se extiende a situaciones en donde el mismo proceso provea de múltiples servicios. Por ejemplo, cuando una sola firma digital provee de autenticación de integridad y de autenticación de origen, o cuando una sola llave simétrica es usada para cifrar y para autenticar en una sola operación.

E.3. Criptoperiodos

Un criptoperiodo es el rango de tiempo durante el cual una llave es válida. Algunas de las funciones de los criptoperiodos son:

- Limitar el total de daños si una sola llave se ve comprometida.
- Limitar el uso de un algoritmo en particular (*particular* en el sentido de la instancia misma del algoritmo).
- Limitar el tiempo disponible para intentos de ataques sobre los mecanismos que protegen a la llave del acceso sin autorización.

- Limitar el periodo en el cual la información puede verse comprometida por revelaciones inadvertidas de llaves a entidades sin autorización.
- Limitar el tiempo disponible para ataque criptoanalíticos.

A continuación se enlistan los principales factores a tomar en cuenta al momento de establecer un criptoperiodo:

- La fuerza del mecanismo criptográfico (el algoritmo, la FUERZA EFECTIVA de la llave, el tamaño de bloque, el modo de operación, etc.).
- El entorno de operacion (p. ej. un lugar de acceso restringido, una terminal pública).
- El volumen de información transmitida, o el número de transacciones.
- La función de seguridad (cifrado de datos, firma digital, derivación de llaves, etc.).
- El método de entrada de la llave (por teclado, a nivel de sistema).
- El número de nodos en una red que comparten la misma llave.
- El número de copias de la llave distribuidas.
- Rotaciones de personal.
- La amenaza de los adversarios (*¿de quién se está protegiendo la información?, ¿cuáles son sus capacidades?*)
- La amenaza de avances tecnológicos (nuevos límites para el problema del logaritmo discreto, computadoras cuánticas).

La duración de los criptoperiodos también debe tomar en cuenta cuáles son los riesgos de las actualizaciones de llaves. En general, entre más cortos sean los criptoperiodos, mejor; sin embargo, si los mecanismos de distribución de llaves son manuales y propensos a errores humanos, entonces el riesgo se invierte. En estos casos (en especial cuando se utiliza CRIPTOGRAFÍA FUERTE) resulta mejor tener pocas y bien controladas distribuciones manuales, a que estas sean frecuentes y mal controladas.

Las consecuencias de una exposición se miden de acuerdo a qué tan sensible es la información, qué tan críticos son los procesos protegidos, y qué tan alto es el costo de recuperación en caso de exposición. La sensibilidad se refiere al periodo de tiempo durante el cual la información debe estar protegida (5 segundos, 5 minutos, 5 horas o 5 años) y a las consecuencias potenciales en caso de pérdida de protección. En general, entre más sensible sea la información protegida, el criptoperiodo debe ser menor (sin llegar a que sea contraproducente).

Tipo de llave	Criptoperiodo	
	Periodo de uso de emisor (PE)	Periodo de uso de receptor (PR)
1.- Llave privada para firma	1 a 3 años	-
2.- Llave pública para verificación de firma	Depende del tamaño de la llave	
3.- Llave simétrica para autenticación	≤ 2 años	$\leq PE + 3$ años
4.- Llave privada para autenticación	1 a 2 años	
5.- Llave pública para autenticación	1 a 2 años	
6.- Llave simétrica para cifrado de datos	≤ 2 años	$\leq PE + 3$ años
7.- Llave simétrica como envoltura	≤ 2 años	$\leq PE + 3$ años
8.- Llave simétrica para generadores aleatorios	ver SP800-90	-
9.- Llave simétrica maestra	alrededor de 1 año	-
10.- Llave privada para transporte	≤ 2 años	
11.- Llave pública para transporte	1 a 2 años	
12.- Llave simétrica para acuerdo	1 a 2 años	
13.- Llave privada para acuerdo	1 a 2 años	
14.- Llave pública para acuerdo	1 a 2 años	
15.- Llave privada efímera para acuerdo	Solo 1 transacción	
16.- Llave pública efímera para acuerdo	Solo 1 transacción	
17.- Llave simétrica para autorización	≤ 2 años	
18.- Llave privada para autorización	≤ 2 años	
19.- Llave pública para autorización	≤ 2 años	

Tabla E.2: Criptoperiodos sugeridos por tipo de llave

Algunos otros factores a tomar en cuenta incluyen el uso de las llaves y el costo de actualizaciones. En cuanto al uso de las llaves, generalmente se hace distinción en cuanto a si la información protegida solamente se está transfiriendo, o si se va a almacenar; en general los criptoperiodos son más largos en caso de almacenamiento, dado el costo que puede representar el recifrado de toda una base de datos. Esto está relacionado con el costo de las actualizaciones: cuando el volumen de información protegida es demasiado grande o cuando esta se encuentra distribuida en distintos puntos geográficos, el costo de un cambio de llaves puede ser demasiado elevado.

La tabla E.2 muestra las sugerencias del NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY (NIST) en [38] en cuanto a la duración de los criptoperiodos según cada tipo de llave.

E.4. Estados de llaves y transiciones

En la figura E.1 se muestra un diagrama de estados con los posibles comportamientos de una llave criptográfica. Un criptoperiodo inicia al entrar en el estado «activo» (transición 4) y termina al llegar al estado «destruido». En el caso de esquemas asimétricos, las transiciones se aplican a ambos pares de llaves. Se debe llevar un registro de la fecha y hora (y en algunos casos de las razones) de cualquier cambio de estado.

Hay varias posibles razones por las que se puede llegar a un estado «suspendido»: la entidad dueña de una firma digital no se encuentra disponible; se sospecha que la integridad de la llave está comprometida, por lo que se pasa a este estado en lo que se investiga más en profundidad; entre otros. Una llave que está en este estado («suspendido») no debe ser usada bajo ninguna circunstancia.

Las llaves que se encuentran en el estado «inactivo» no deben ser usadas para aplicar protección criptográfica, pero en algunos casos, pueden ser usadas para procesar información protegida con criptografía. Por ejemplo, las llaves simétricas usadas para autenticación, cifrado de datos o envoltura de llaves, pueden ser usadas para procesar información hasta que acabe el periodo del emisor de la llave emisora.

El estado «comprometido» representa a las llaves a las que una entidad sin autorización tiene o ha tenido acceso. Las llaves comprometidas no deben ser usadas para aplicar protección criptográfica, sin embargo, en algunos casos es posible que sean usadas para procesar información (en condiciones sumamente controladas); por ejemplo, si la cierta información fue firmada antes de que se produjera la brecha de seguridad, las llaves pueden ser usadas para validar esta firma.

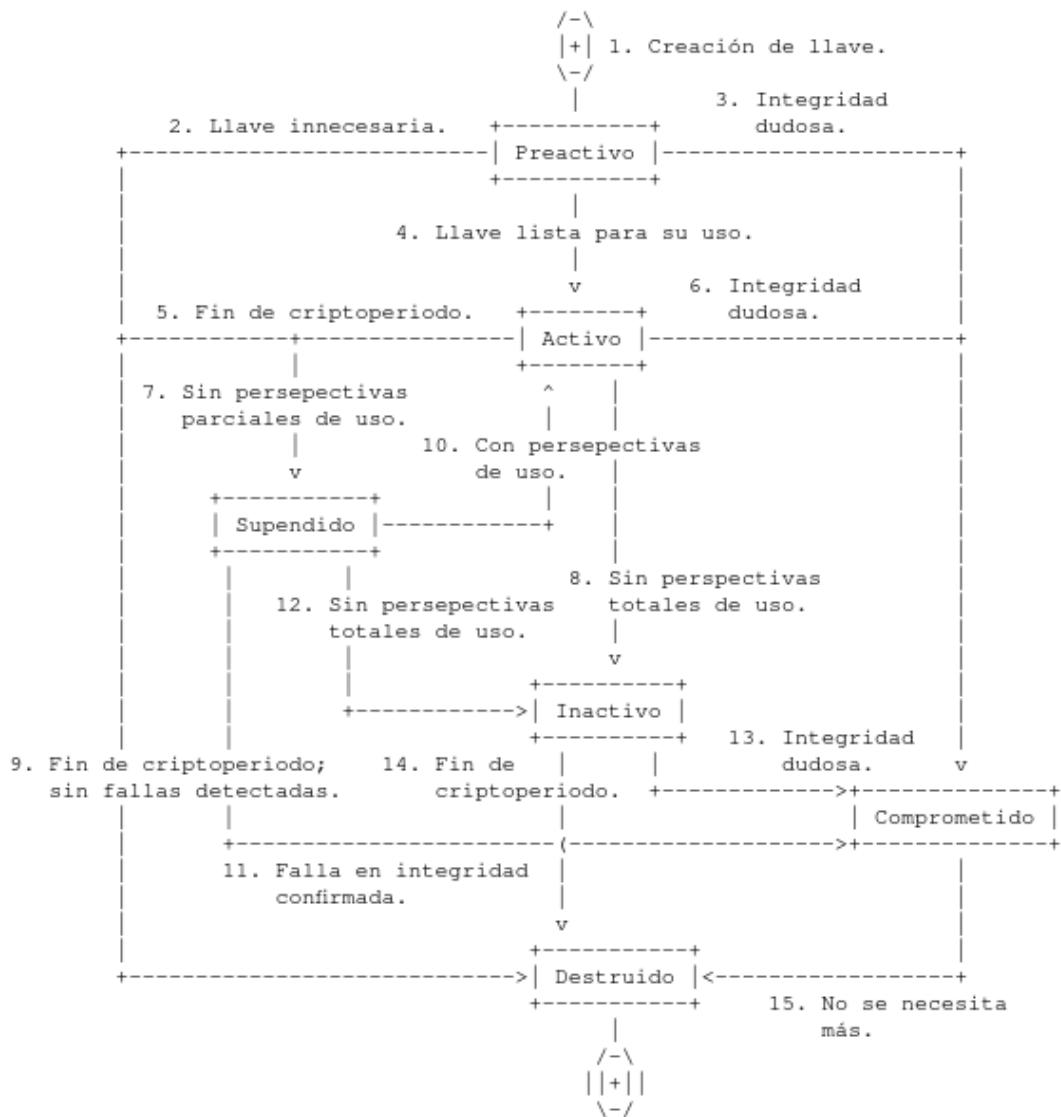


Figura E.1: Diagrama de estado de llaves criptográficas.

Apéndice F

Páginas estáticas

«Our intellectual powers are rather geared to master static relations and that our powers to visualize processes evolving in time are relatively poorly developed. For that reason we should do (as wise programmers aware of our limitations) our utmost to shorten the conceptual gap between the static program and the dynamic process, to make the correspondence between the program (spread out in text space) and the process (spread out in time) as trivial as possible.»

EDSGER DIJKSTRA.

En este apéndice se muestran los contenidos de las páginas estáticas de la aplicación web del servicio tokenizador: la página de inicio y la página de documentación. El contenido de estas dos páginas está regido por los requerimientos REQFAPI-22 MOSTRAR PROMOCIÓN DEL SERVICIO TOKENIZADOR y REQFAPI-23 MOSTRAR DOCUMENTACIÓN DEL SERVICIO TOKENIZADOR, respectivamente. Estas páginas se encuentran disponibles en:

- [HTTPS://RICARDO-QUEZADA.159.65.96.59.xip.io/LOVELACE/](https://RICARDO-QUEZADA.159.65.96.59.xip.io/LOVELACE/)
- [HTTPS://RICARDO-QUEZADA.159.65.96.59.xip.io/LOVELACE/DOCUMENTACION](https://RICARDO-QUEZADA.159.65.96.59.xip.io/LOVELACE/DOCUMENTACION)

Para evitar el duplicado de información se ocuparon las siguientes dependencias. La fuente de información original es el archivo .tex, mientras que el que ocupa la aplicación web es un .html dependiente del primero.

DEPEST-01 Traductor entre lenguajes de marcado: Pandoc.

Licencia: GPL v2

Página oficial: <https://github.com/jgm/pandoc>

Permite hacer conversiones entre archivos de marcado. En este caso, se ocupa para pasar de un archivo fuente de LATEX a un HYPERTEXT MARKUP LANGUAGE (HTML).

DEPEST-02 Filtro para archivos de LATEX: Pandoc-latex-levelup.

Licencia: BSD

Página oficial: <https://github.com/daamien/pandoc-latex-levelup>

Filtro para DEPEST-01 TRADUCTOR ENTRE LENGUAJES DE MARCADO: PANDOC. Permite hacer corrimientos entre la profundidad de los archivos de origen y los archivos destino.

F.1. Página de inicio

F.1.1. ¿Qué es la tokenización?

Primero, ¿qué es un token?

Antes de definir la *tokenización*, es menester definir lo que es un token: normalmente, se le conoce como un valor representativo de otro valor; el concepto de token es ampliamente utilizado en otros contextos, y, dependiendo de este, varían sus características. En la criptografía (que es el contexto en el que se encuentra este servicio), este *valor representativo* no debe tener una relación directa con el valor original; de hecho, que un adversario posea un token no representa peligro alguno, pues no es posible, sin los mecanismos de detokenización, regresar al valor original. Por lo tanto, un token puede ser utilizado para proteger información confidencial.

Sobre la tokenización

Con esta definición de un token, se puede definir a la tokenización como el proceso en el que se obtiene el valor sustituto. Este paradigma de sustituir datos sensibles por datos representativos (y sin valor) es relativamente nuevo, pero muy útil al momento de proteger la información, pues permite concentrar toda la información valiosa en un solo lugar y dejar de preocuparse por protegerla en todos lados (en vez de vigilar un castillo, se vigila una habitación).

La operación inversa, es decir dado un token, obtener el PAN (número de cuenta, por sus siglas en inglés) que le corresponde, es llamada *detokenización*.

F.1.2. ¿Por qué utilizar un servicio de tokenización?

Desde que el comercio electrónico comenzó a popularizarse, los fraudes relacionados con las tarjetas bancarias y el comercio en línea crecieron, pues los sitios bancarios y de comercio no estaban preparados para la demanda y las amenazas de seguridad a las que comenzaron a enfrentarse. Por lo tanto, en 2004, fue publicado por primera vez un estándar (PCI DSS, respaldado por compañías como VISA y MasterCard) donde se dan las guías de seguridad que deberían seguir todos aquellos que traten con datos de tarjetas o realicen transacciones bancarias.

Aunque este estándar no es obligatorio para aquellas entidades que realicen menos de 20 000 transacciones en un año; es extremadamente recomendable que sigan la guía para mantener segura la información y transacciones. Empero, satisfacer completamente el estándar es una verdadera proeza, cuenta con más de 200 requerimientos para asegurar la robustez y seguridad del sistema que trata con los datos y pagos con tarjetas bancarias.

Es aquí donde entran los servicios de tokenización, pues se encargan de cumplir con todos los requerimientos y quitarle esa pesada carga para que pueda enfocarse en sus actividades sin preocuparse por el estándar y sus implementaciones. Obviamente, es muy importante saber cómo es que servicio de tokenización está obteniendo los tokens, pues no basta con asegurar que son seguros; los servicios deben indicar (y hacer públicos) los algoritmos que utilizan para generar los tokens. Por lo tanto, que un servicio clame que tiene la mejor manera para generar tokens, pero no especifique cómo es que los está obteniendo (o que lo haga de manera vaga, por ejemplo, asegurando que sus tokens son obtenidos aleatoriamente sin dar más detalles), es inadmisible, pues un buen algoritmo tokenizador no depende en absoluto de ser mantenido en secreto.

F.1.3. ¿Por qué utilizar ESTE servicio de tokenización?

A diferencia de otros competidores, nosotros somos transparentes respecto a la manera en la que se generan los tokens (¡nuestras implementaciones son públicas!); actualmente, hay cinco algoritmos distintos para obtener tokens y usted puede decidir, para cada token, el algoritmo que será utilizado: un poco más adelante hay una sección que explica brevemente cada algoritmo.

F.1.4. ¿Qué es el PCI SSC?

El PAYMENT CARD INDUSTRY SECURITY STANDARDS COUNCIL (PCI SSC) es una organización internacional encargada de estandarizar, desarrollar e informar sobre cómo hacer transacciones bancarias seguras.

Esta organización desarrolló el PCI Data Security Standard (DSS): un estándar en el que se indica la manera de mantener la información bancaria segura mediante la implementación de protocolos de seguridad.

F.1.5. Sobre los algoritmos de tokenización

Nuestro sistema permite seleccionar el algoritmo con el que se desea tokenizar un dato. Se tienen dos tipos de algoritmos: los reversibles y los irreversibles. La diferencia entre unos y otros radica en la operación de detokenización, pues los primeros (los reversibles) *descifran* el token para obtener el PAN; mientras que los segundos hacen una consulta a una base de datos para obtener el PAN.

Contamos con 2 algoritmos de tokenización reversibles: FFX y BPS, y 3 algoritmos de tokenización irreversibles: TKR, AHR y DRBG. A continuación se encuentra una breve descripción de cada uno.

FFX

Este es un cifrado que permite cifrar cadenas de cualquier tamaño, compuestas por cualquier tipo de caracteres, el cual es usado para tokenizar. Fue publicado por Mihir Bellare, Phillip Rogaway y Terence Spies en 2009, y, en 2016, el National Institute of Standards and Technology (NIST) le dio el nombre de FF1 a este algoritmo.

De forma general, el algoritmo usa redes Feistel junto con primitivas criptográficas (funciones hash o cifrados por bloques) adaptadas en la función de ronda de la red para lograr preservar el formato del dato dado para tokenizar; esto significa que el token tendrá la misma longitud y se mantendrá en el mismo dominio que el dato original.

Para conocer más sobre este algoritmo revise el siguiente artículo: THE FFX MODE OF OPERATION FOR FORMAT-PRESERVING ENCRYPTION.

BPS

Es un cifrado que preserva el formato usado para tokenizar, propuesto por Eric Brier, Thomas Peyrin y Jacques Stern en 2010 y renombrado por el NIST como FF3.

Al igual que FFX, usa redes Feistel y primitivas criptográficas para lograr que el token tenga el mismo formato que el dato original.

Para conocer más sobre este algoritmo revise el siguiente artículo: BPS: A FORMAT-PRESERVING ENCRYPTION PROPOSAL.

TKR

Propuesto por Sandra Díaz Santiago, Lil María Rodríguez Henríquez y Debrup Chakraborty en 2016, es un algoritmo que usa primitivas criptográficas para generar tokens aleatorios y almacenarlos en una base de datos segura para guardar su relación con el dato original.

Para conocer más sobre este algoritmo revisa el artículo: A CRYPTOGRAPHIC STUDY OF TOKENIZATION SYSTEMS.

AHR

Algoritmo de tokenización publicado por Riccardo Aragona, Riccardo Longo y Massimiliano Sala en 2017.

Este se basa en un cifrado de bloques y una caminata cíclica para generar tokens que mantengan el formato; los tokens son almacenados en una base de datos segura para guardar la relación con los datos originales.

Para conocer más sobre este algoritmo revise el siguiente artículo: SEVERAL PROOFS OF SECURITY FOR A TOKENIZATION ALGORITHM.

DRBG

La tokenización por este medio se logra produciendo una cadena de bits aleatoria con un DRBG que se interpreta de forma especial para que tenga el formato del dato original.

El DRBG puede estar basado tanto en funciones hash como en cifrados por bloque.

Para conocer más sobre este algoritmo revise el siguiente artículo: NIST SPECIAL PUBLICATION 800-90A REVISION 1.

F.1.6. Una comparación

En aras de arrojar luz sobre el desempeño de los algoritmos, a continuación se encuentran dos gráficas que muestran el tiempo que les toma a sendos algoritmos realizar el mismo número de tokenizaciones. Naturalmente, los algoritmos irreversibles son más lentos, pues deben realizar operaciones en la base de datos; por lo que se comparan algoritmos reversibles con algoritmos reversibles y algoritmos irreversibles con algoritmos irreversibles.

En esta primera gráfica se comparan los tiempos de tokenización y detokenización de los algoritmos de FFX y BPS mientras el número de operaciones se incrementa.

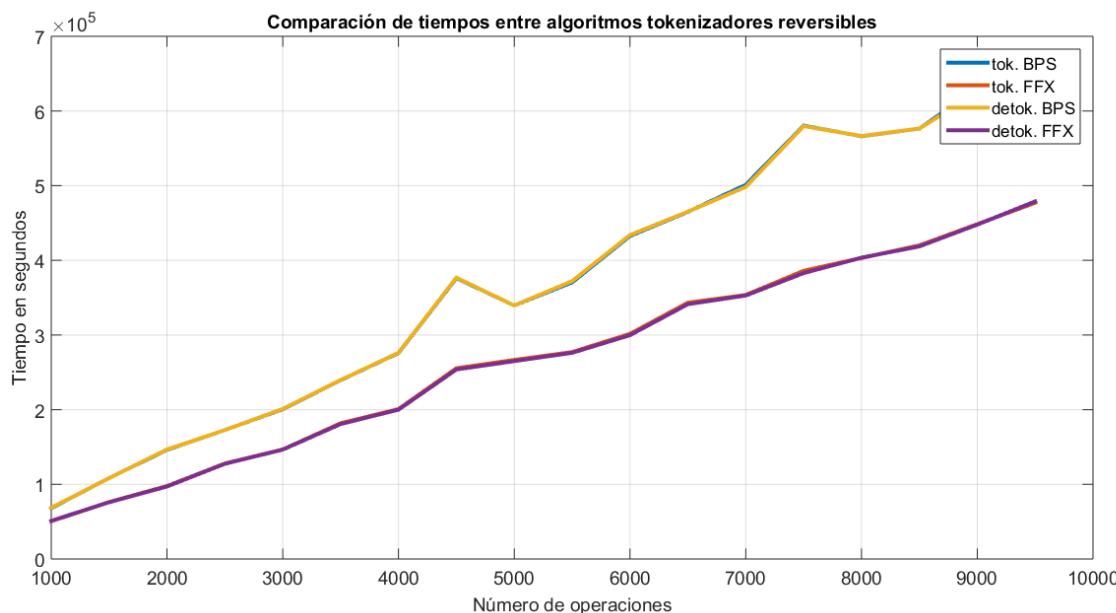


Figura F.1: Comparación de algoritmos reversibles

Es importante observar que los tiempos de tokenización y detokenización son tan parecidos, que sus gráficas se sobreponen completamente.

En esta otra se muestra los tiempos de tokenización y detokenización de TKR, AHR y DRGB mientras el número de operaciones se incrementa.

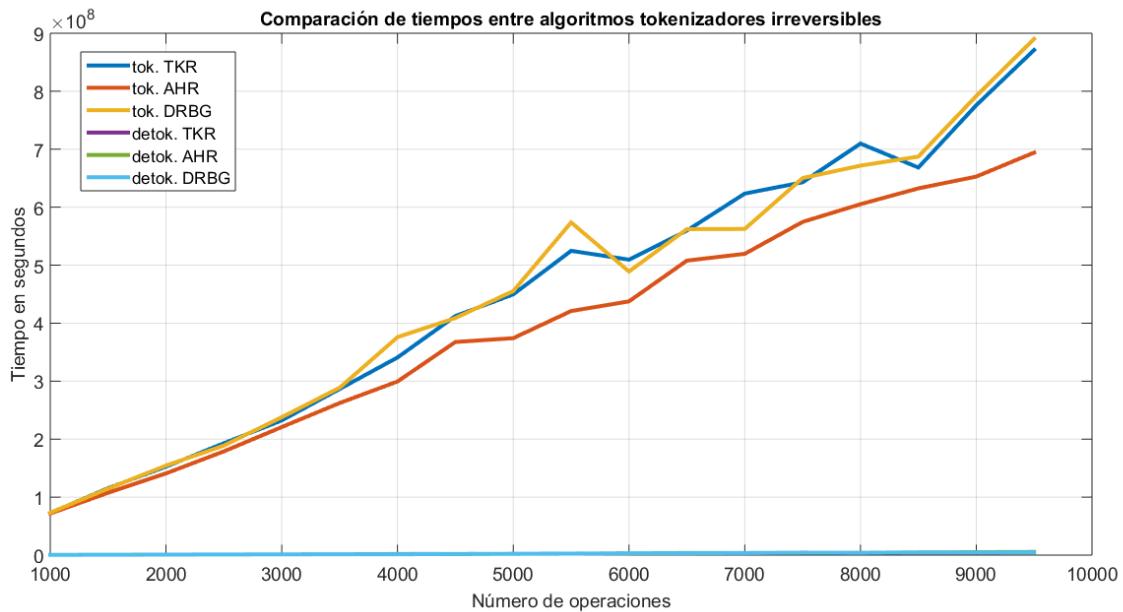


Figura F.2: Comparación de algoritmos irreversibles

Aquí es importante resaltar que, si bien el tiempo de tokenización es considerable (comparándolo con los reversibles), el tiempo de detokenización es mucho más corto, incluso es menor a los algoritmos reversibles.

F.2. Página de documentación

F.2.1. Cómo utilizar la API web

La API permite realizar tres operaciones que se explican a continuación; todas las operaciones se deben realizar mediante peticiones HTTP utilizando el método POST. Todas las peticiones deben tener los encabezados de tipo de contenido (especificando que se está enviando un JSON) y el encabezado de autenticación básica (siguiendo el formato dado en el RFC 7617). En el cuerpo de la petición se especifica el método y el PAN o el método y el token.

Tokenización

Permite, a partir del número de una tarjeta, obtener un token asociado a él. Debe encontrarse en estado **aprobado** o en **cambio de llaves**; en todos los casos, la llave que se utilizará al tokenizar siempre será la que tenga estado **actual**.

Uso y ejemplo En el cuerpo de la petición se debe señalar el **PAN** y el **método**, siguiendo el siguiente esquema:

```
POST /programa_tokenizador/tokenizar HTTP/1.1
{
    Authorization: Basic QWxhZgRpbjpPcGVuU2VYW11
    Content-Type: Application/JSON
}
{
    "pan" : "28045869693113314",
    "metodo" : "FFX"
}
```

Los valores para el campo método son: FFX, BPS, TKR, AHR o DRBG.

Errores A continuación se listan los errores que pueden surgir al tokenizar un PAN.

- 400 - **El PAN recibido es inválido.**

El PAN enviado no es válido; para serlo debe tener una longitud de 12 a 19 dígitos y el último dígito debe ser calculado mediante el algoritmo de Luhn.

- 400 - **Ya existe un token asociado, utilice la función de retokenización.**

El PAN que fue enviado ya tiene un token asignado, pero se encuentra en cambio de llaves; retokenice este token.

- 401 - **Es necesario proveer credenciales para realizar esta operación**

Este error surge cuando el encabezado de autenticación no se ha encontrado.

- 401 - **Las credenciales son incorrectas**

Este error surge cuando las credenciales especificadas en el encabezado no se encuentran registradas en el sistema.

- 403 - El usuario no tiene los privilegios para esta operación

Aunque el usuario está registrado, no tiene los permisos necesarios para realizar esta operación.

- 403 - EL usuario no se encuentra en el estado necesario para esta operación

Aunque el usuario está registrado, no se encuentra en el estado requerido (**aprobado** o **en cambio de llaves**) para realizar esta operación; podría aún no estar aprobado o haber sido rechazado o puesto en la lista negra.

- 403 - [Token]

Ya existe un token asociado al PAN que envió para tokenizar; por regla, solo se puede tener uno, así que se regresa el token que fue creado para este PAN. Si se encuentra **en cambio de llaves**, y el token ya fue retokenizado, regresa el más nuevo.

- 403 - Parámetros incompletos o incorrectos

El servidor no pudo encontrar el parámetro **pan**, **metodo** o ambos.

Detokenización

Permite, a partir de un token, obtener el PAN asociado a él. Debe encontrarse en estado **aprobado** o **en cambio de llaves**; el segundo estado se alcanza cuando se inicia el refresco de llaves desde el sitio web y, en caso de haberlo iniciado, es menester especificar si se desea detokenizar utilizando la llave **anterior** o la llave que fue creada al iniciar el refresco de llaves (la llave **actual**). Si no es especificado, se utiliza la llave **actual**.

Uso y ejemplo En el cuerpo de la petición se debe señalar el **PAN** y el **método**, siguiendo el siguiente esquema:

```
POST /programa_tokenizador/detokenizar HTTP/1.1
{
    Authorization: Basic QWxhZgRpbjpPcGVuU2VYW11
    Content-Type: Application/JSON
}
{
    "token" : "28045869693113314",
    "metodo" : "FFX"
    "versionLlave" : "anterior"
}
```

Los valores para el campo **método** son: FFX, BPS, TKR, AHR o DRBG. Los valores para el campo de **versiónLlave** son: anterior y actual.

Errores A continuación se listan los errores que pueden surgir al detokenizar un token.

- 400 - El token recibido es inválido

El servidor recibió un token que no es válido o generado por este servicio.

- 400 - El token no existe en la base de datos.

El token, aunque válido, no se encuentra registrado y asociado con el cliente que envió la petición.

- 401 - Es necesario proveer credenciales para realizar esta operación

Este error surge cuando el encabezado de autenticación no se ha encontrado.

- 401 - Las credenciales son incorrectas

Este error surge cuando las credenciales especificadas en el encabezado no se encuentran registradas en el sistema.

- 403 - El usuario no tiene los privilegios para esta operación

Aunque el usuario está registrado, no tiene los permisos necesarios para realizar esta operación.

- 403 - El usuario no se encuentra en el estado necesario para esta operación

Aunque el usuario está registrado, no se encuentra en el estado requerido (aprobado o en cambio de llaves) para realizar esta operación; podría aún no estar aprobado o haber sido rechazado o puesto en la lista negra. Ingrese al sitio web para obtener más información sobre su estado.

- 403 - Parámetros incompletos o incorrectos

El servidor no pudo encontrar el parámetro `token`, `método` o ambos.

Retokenización

Permite, a partir de un token, obtener la nueva versión del mismo. Debe encontrarse en estado **en cambio de llaves** y proveer en el cuerpo de la petición el token que se desea actualizar; para obtener esta nueva versión del token, se detokenizará con la llave anterior y el resultado se tokenizará con la llave actual. Es menester utilizar el mismo método con el que fue creado el token para realizar esta operación.

Uso y ejemplo En el cuerpo de la petición se debe señalar el **token** y el **método**, siguiendo el siguiente esquema:

```
POST /programa_tokenizador/retokenizar HTTP/1.1
```

```
{
```

```
  Authorization: Basic QWxhZgRpbjpPcGVuU2VYW11
```

```
  Content-Type: Application/JSON
```

```
}

{
    "token" : "28045869693113314",
    "metodo" : "FFX"
}
```

Los valores para el campo método son: FFX, BPS, TKR, AHR o DRBG.

Errores A continuación se listan los errores que pueden surgir al detokenizar un token.

- 400 - El token recibido es inválido

El servidor recibió un token que no es válido o generado por este servicio.

- 400 - El token no existe en la base de datos

El token, aunque válido, no se encuentra registrado y asociado con el cliente que envió la petición.

- 401 - Es necesario proveer credenciales para realizar esta operación

Este error surge cuando el encabezado de autenticación no se ha encontrado.

- 401 - Las credenciales son incorrectas

Este error surge cuando las credenciales especificadas en el encabezado no se encuentran registradas en el sistema.

- 403 - El usuario no tiene los privilegios para esta operación

Aunque el usuario está registrado, no tiene los permisos necesarios para realizar esta operación.

- 403 - EL usuario no se encuentra en el estado necesario para esta operación

Aunque el usuario está registrado, no se encuentra en el estado requerido (en cambio de llaves).
Ingrese al sitio web para obtener más información sobre su estado.

- 403 - Parámetros incompletos o incorrectos

El servidor no pudo encontrar el parámetro `token`, `metodo` o ambos.

- 403 - [Token]

Está intentando retokenizar un token que ya fue retokenizado. El servidor regresa el token creado más recientemente.

Apéndice G

Sobre este documento

*«The trouble with computer science today
is an obsessive concern with form instead
of content.»*

Form and Content in Computer Science,
1970 ACM Turing Lecture, MARVIN
MINSKY.

En este apéndice se discuten algunos de los aspectos más relevantes de la elaboración de este documento. Primero, se presentan algunas de las dependencias más importantes de L^AT_EX; después, se presenta una lista de los comandos hechos específicamente para este reporte y se muestran, a modo de ejemplo, algunos de ellos.

Primero, ¿por qué L^AT_EX? Una de las principales ventajas con las que se promociona a L^AT_EX es la separación de responsabilidades: el autor de un documento solamente debe de preocuparse por el contenido, mientras que el programa, L^AT_EX, es el responsable de la forma, esto es, de la apariencia final del documento [70]. En el caso de este trabajo, lo anterior no es enteramente cierto, pues hay un alto grado de modificaciones sobre los comportamientos por defecto (de lo contrario no existiría este apéndice). Lo que sí es cierto es que el uso de L^AT_EX implica una separación base entre la forma y el contenido de una trabajo y, siendo cuidadosos, esta separación se puede mantener aún cuando se modifique el comportamiento por defecto.

La segunda razón de peso tiene que ver con las circunstancias del producto requerido: se trata de un documento muy extenso que requiere modificaciones simultáneas de todo el equipo; en particular, se necesita que las modificaciones al documento estén dentro del seguimiento del control de versiones (dependencia DEPPT-05 CONTROL DE VERSIONES: GIT v2.17). A diferencia de un procesador de textos WHAT YOU SEE IS WHAT YOU GET (WYSIWYG), L^AT_EX trabaja con archivos de texto plano, lo cual permite mantener el control de versiones tan claro como se quiera. El punto anterior también permite importar buenas prácticas de programación en la elaboración del documento: la modularización del código en archivos y carpetas significativos (este documento está distribuido en alrededor de 500 archivos, la estructura de los archivos emula a la estructura del documento); líneas cortas (80 caracteres en este caso) para facilitar la lectura directa en archivos fuente y para facilitar la comparación de versiones.

La inclusión de este apéndice es una forma de reconocimiento hacia el tiempo y esfuerzo dedicados para que este documento llegara a ser lo que es.

G.1. Dependencias

El documento depende actualmente de 138 paquetes de L^AT_EX; aquí se enlistan aquellos cuyo uso responde a un proceso de decisión, y no a una simple necesidad.

DEPTEX-01 Elaboración de diagramas: TikZ.

Licencia: LPPL v1.3

Página oficial: [HTTPS://WWW.CTAN.ORG/PKG/PGF](https://www.ctan.org/pkg/pgf)

Paquete para crear gráficos. La interfaz al usuario es similar a la de L^AT_EX: define un lenguaje para la descripción del gráfico; el producto final se obtiene mediante un proceso de compilado.

Existen varias ventajas al utilizar TikZ. Las dos principales son las mismas que las del propio L^AT_EX: al trabajar con texto plano, los diagramas se encuentran dentro del seguimiento del control de versiones;

con un poco de cuidado, se puede mantener separada a la forma del contenido, lo cuál permite crear diagramas con exactamente el mismo aspecto. Otra ventaja es la integración entre los diagramas y el propio documento: TikZ produce comandos en PROGRESSIVE GRAPHICS FILE (PGF) que se encuentran incrustados directamente en el PORTABLE DOCUMENT FORMAT (PDF) producto, lo cual implica que el texto de los diagramas es seleccionable (en algunos diagramas incluso hay hipervínculos, por ejemplo, el diagrama de estados 6.1); la tipografía es la misma para diagramas y documento. La única desventaja está dada por el tiempo de aprendizaje de la sintaxis, que en comparación con un editor gráfico, es algo mucho más lento. A modo de ejemplo, el código fuente 47 muestra un fragmento del archivo fuente del diagrama 2.9

```

1 %
2 % Operación de addRoundKey en AES,
3 % Capítulo de marco teórico.
4 % Proyecto Lovelace.
5 %
6
7 \begin{tikzpicture}[
8   celda/.style = {
9     rectangle,
10    draw = black,
11    fill = white,
12    thin,
13    inner sep = 2mm,
14    align = center,
15    minimum height = 10mm,
16    text width = 6mm}]
17
18 \begin{scope}[
19   local bounding box = caja_a]
20   \foreach \x in {0,...,3}
21     \foreach \y in {0,...,3}
22       {\node[celda]
23         (a_\y_\x)
24         at (\x cm, \y cm * -1cm)
25         {\a_{\y,\x}};}
26   \node[celda, thick,
27     fill = gray!50!white]
28     (a_2_2)
29     at (2, -2)
30     {\a_{2, 2}};
31 \end{scope}
32
33 \begin{scope}[
34   local bounding box = caja_b,
35   shift={(\x(caja_a.north east) + (5cm, -0.5cm))}]
36   \foreach \x in {0,...,3}
37     \foreach \y in {0,...,3}
38       {\node[celda]
```

Código fuente 47: Ejemplo de diagrama en TikZ; diagrama 2.9

DOCUMENTOS_ENTREGABLES/REPORTE_TECNICO/CONTENIDOS/MARCO_TEORICO/BLOQUES/DIAGRAMAS/
ADDRONDKEY.TIKZ.TEX

DEPTEX-02 Diagramas UML: TikZ-UML.

Licencia: THE LATEX PROJECT PUBLIC LICENSE (LPPL)

Página oficial: [HTTP://PERSO.ENSTA-PARISTECH.FR/~KIELBASI/TIKZUML/INDEX.PHP](http://perso.ensta-paristech.fr/~kielbasi/tikzuml/index.php)

Conjunto de comandos alrededor de DEPTEX-01 ELABORACIÓN DE DIAGRAMAS: TikZ para la creación de diagramas en UNIFIED MODELING LANGUAGE (UML).

Define una sintaxis que permite crear diagramas de TikZ pensando solamente en el contenido, sin tener que preocuparse por el aspecto. El código fuente 48 muestra un fragmento del diagrama de secuencia 6.22.

```
1 %
2 % Diagrama de secuencia de registro de cliente,
3 % Análisis y diseño de aplicación web,
4 % Reporte técnico.
5 % Proyecto Lovelace.
6 %
7
8 \begin{tikzpicture}[]
9   \begin{umlseqdiag}
10
11     % Líneas de vida
12     \umlactor[class = Usuario]{visitante}
13     \umlboundary[no ddots]{IUAPI-01}
14     \umlcontrol[class = Servicio]{servicioGeneral}
15     \umlentity[class = Negocio]{negocioGeneral}
16     \umldatabase[no ddots]{Modelo}
17
18     % Presionar botón
19     \begin{umlcall}[%]
20       op = {Botón «Registrar»},%
21       return = IUAPI-04{visitante}{IUAPI-01}
22     \end{umlcall}
23
24     % Entrada de correo
25     \begin{umlcall}[%]
26       op = {Correo electrónico},
27       return = resultado]{visitante}{IUAPI-01}
28     \begin{umlcall}[%]
29       op = {validarCorreo(correo)}]{IUAPI-01}{IUAPI-01}
30     \end{umlcall}
31   \end{umlseqdiag}
```

Código fuente 48: Ejemplo de diagrama en TikZ-UML; diagrama 6.22.

DOCUMENTOS_ENTREGABLES/REPORTE_TECNICO/CONTENIDOS/ANALISIS_Y_DISEÑO_API_WEB/DISEÑO/
DIAGRAMAS/SECUENCIA_REGISTRAR_CLIENTE.TIKZ.TEX

DEPTEX-03 Para distribución de código fuente: Import.

Licencia: PDS

Página oficial: [HTTPS://WWW.CTAN.ORG/PKG/IMPORT](https://www.ctan.org/pkg/import)

Paquete para importar archivos fuente de forma relativa al archivo fuente actual.

Si se trabaja con documentos pequeños, la importancia de este paquete no es muy clara; sin embargo, al trabajar con un documento distribuido en más de 500 archivos, los importados relativos son indis-

pensables. Además de que, si se trabajara con el comportamiento por defecto (importados absolutos al archivo origen) las rutas internas de los archivos más profundos serían ridículamente largas e inmanejables, el importado relativo también permite mover carpetas enteras sin tener que alterar el código interno. Por ejemplo, para cambiar una sección de capítulo basta con mover la carpeta correspondiente de lugar y cambiar la línea de llamada. En el código fuente 49 se muestra un fragmento del archivo principal del documento en donde se hace uso de este paquete.

```

27 \setcounter{tocdepth}{5}
28 \newpage
29
30 \import{reporte_tecnico/contenidos/}{simbologia.tex}
31
32 \mainmatter
33 \import{reporte_tecnico/contenidos/intro/}{intro.tex}
34 \import{reporte_tecnico/contenidos/marco_teorico/}{marco_teorico.tex}
35
36 \parte{Generación de tokens}{parte:tokens}{%
37   \epigrafe{%
38     ``Begin at the beginning,'' the King said, very gravely, ``and go on till
39     you came to the end: then stop.''}{%
40       \textsc{Lewis Carroll}.}}
41

```

Código fuente 49: Muestra de importados relativos.

DOCUMENTOS_ENTREGABLES/REPORTE_TECNICO/REPORTE_TECNICO.TEX

DEPTEX-04 Para administración de fuentes bibliográficas: BibIATEX y Biber.

Licencia: LPPL y PAL

Página oficial: <https://www.ctan.org/pkg/biber>

BibIATEX es el sucesor de BibTEX, el gestor de fuentes bibliográficas por defecto de IATEX. BibIATEX es la interfaz para Biber: un programa en Perl que se encarga de la construcción de la bibliografía.

En el archivo fuente 50 se muestran un par de las referencias de este documento ([49] y [65])¹. Existen varias ventajas al trabajar de esta forma. Primero, al igual que lo que pasa al trabajar con otros paquetes y con IATEX mismo, la forma y el contenido son independientes entre sí (véase ACOPLAMIENTO); el contenido es el que se muestra en el fragmento de código 50, mientras que la forma va dada por la configuración de BibIATEX. Por ejemplo, las referencias en este trabajo se encuentran en formato INSTITUTE OF ELECTRICAL AND ELECTRONIC ENGINEERS (IEEE), sin embargo, en el supuesto de que se quisieran cambiar a AMERICAN PSYCHOLOGICAL ASSOCIATION (APA), bastaría con dar la orden al programa, sin tener que cambiar las descripciones fuente. Otra ventaja es la reutilización de las fuentes a través de distintos documentos: el mismo archivo de referencias se usa en este documento, en las presentaciones de Trabajo Terminal, en el artículo sobre tokenización, etcétera.

¹DIGITAL BIBLIOGRAPHY AND LIBRARY PROJECT (DBLP) es una base de datos en línea ([HTTP://DBLP.ORG/](http://dblp.org/)) enfocada a documentos de ciencias de la computación. Muchas de las fuentes de este trabajo provienen de esta herramienta.

```
899 @article{DBLP:journals/rfc/rfc4648,
900   author    = {Simon Josefsson},
901   title     = {The Base16, Base32, and Base64 Data Encodings},
902   journal   = {{RFC}},
903   volume    = {4648},
904   pages     = {1-18},
905   year      = {2006},
906   url       = {https://doi.org/10.17487/RFC4648},
907   doi       = {10.17487/RFC4648},
908   timestamp = {Tue, 28 Nov 2017 13:58:46 +0100},
909   biburl   = {https://dblp.org/rec/bib/journals/rfc/rfc4648},
910   bibsource = {dblp computer science bibliography, https://dblp.org}
911 }
912
913 @book{DBLP:books/crc/AU1992,
914   author    = {Alfred V. Aho and
915               Jeffrey D. Ullman},
916   title     = {Foundations of Computer Science, {C} Edition},
917   publisher = {Computer Science Press / W. H. Freeman},
918   year      = {1992},
919   url       = {http://i.stanford.edu/\%7Eullman/focs.html},
920   isbn      = {0716782332},
921   timestamp = {Tue, 16 May 2017 17:36:01 +0200},
922   biburl   = {https://dblp.org/rec/bib/books/crc/AU1992},
923   bibsource = {dblp computer science bibliography, https://dblp.org}
924 }
```

Código fuente 50: Muestra de fuentes bibliográficas.

DOCUMENTOS_ENTREGABLES/REFERENCIAS.BIB

G.2. Comandos propios

Como parte de la elaboración de este documento se crearon los siguientes comandos y entornos:

- **\capítulo** Para insertar capítulos con epígrafes.
- **\codigoFuente** Construcción alrededor de *minted* para insertar código fuente.
- **\dependencia** Para definir una dependencia; por ejemplo, las de la sección anterior.
- **\epigrafe** Construcción alrededor del paquete *epigraph* para definir un formato propio.
- **\expresionRegular** Para insertar una expresión regular desde un archivo externo.
- **\formatearNúmero** Para colocar ceros a la izquierda de los comandos que ocupan contadores.
- **\hipervinculo** Para definir vínculos referenciables con una etiqueta en los cuales el texto que aparece en el vínculo queda definido por el origen. La gran mayoría de los vínculos EN ESTE TIPO DE LETRA fueron creados con este comando.
- **\parrafoConListaCorta** Entorno para definir un párrafo con una lista contigua relativamente corta. No hay saltos de línea entre el párrafo y toda la lista.

- **\parrafoConListaLarga** Entorno para definir un párrafo con una lista contigua. No hay saltos de línea entre el párrafo dado y el primer elemento de la lista.
- **\parte** Para insertar partes con epígrafes.
- **\portada** Para definir el formato de la portada del trabajo.
- **\pseudocodigo** Construcción alrededor de *listings* para la inserción de pseudocódigo.
- **\unidadIrrompible** Entorno para evitar saltos de página.
- Comandos de los capítulos de análisis y diseño:
 - **\casoDeUso** Para definir el formato de un caso de uso. Ocupa otros subcomandos, propios del contexto de su caso de uso.
 - **\etiquetaLocal** Define etiquetas con un prefijo único.
 - **\hipervinculoLocal** Para hacer referencia a una trayectoria alternativa mostrando el nombre definido por esta.
 - **\referenciaLocal** Para hacer referencia a un paso de la trayectoria principal.
 - **\trayectoriaAlternativa** Entorno para enlistar los pasos de una trayectoria alternativa.
 - **\trayectoriaPrincipal** Entorno para enlistar los pasos de la trayectoria principal.
 - **\interfazDeUsuario** Para insertar el conjunto de capturas que forman la figura de una interfaz de usuario.
 - **\mensaje** Para insertar un mensaje desde un archivo externo.
 - **\reglaDeNegocio** Para la definición de reglas de negocio.
 - **\requerimiento** Para la definición de requerimientos. Contiene comandos para la definición de *subrequerimientos* y *subsubrequerimientos*.

Es mediante la creación de estos comandos como se logra la separación de forma y contenido de la que se habló anteriormente: el comando define la forma, mientras que el contenido queda definido por el código dependiente del comando. Para exponer esto se muestra, en los códigos fuente 51 y 52 la forma y el contenido, respectivamente, de las dependencias. Estos comandos se utilizan en las secciones de tecnologías de los capítulos de implementaciones, y en la sección anterior de este apéndice.

Los casos de uso son, probablemente, los comandos más complejos; principalmente por el número de subcomandos y entornos asociados. Como ejemplo, el código fuente 53 muestra la definición del caso de uso CUAPI-09 ELIMINAR CUENTA DE UN CLIENTE. El primer argumento define la etiqueta para referirse a este caso de uso: el código `\hipervinculo{cu:eliminar_cliente}` dará como resultado un vínculo como el del final de la oración anterior. El siguiente argumento define el nombre del caso de uso. El primer párrafo del tercer argumento determina la descripción del caso de uso; después de esto van los entornos de la trayectoria principal y las trayectorias alternativas.

```
1 %
2 % Plantilla para dependencias
3 % Proyecto Lovelace.
4 %
5
6 % Contador de dependencias:
7 \newcounter{contador_dependencias}[chapter]
8
9 % Comando para definir una nueva dependencia:
10 % 1. Etiqueta (opcional).
11 % 2. Nombre de categoría.
12 % 3. Nombre de dependencia (con versión).
13 % 4. Licencia.
14 % 5. URL de página oficial.
15 % 6. Descripción.
16
17 \newcommand{\dependencia}[6]
18 [dep:\formatearnumero{contador_dependencias}]{%
19   % Configuración
20   \refstepcounter{contador_dependencias}%
21   \setlength{\separacionAnterior}{\parskip}%
22   \setlength{\parskip}{0em}%
23   \renewcommand{\etiqueta}{%
24     {DEP}\prefijo-\formatearnumero{contador_dependencias}~#2:~#3}%
25   % Código efectivo
26   \pdfbookmark[3]{\etiqueta}{#1}%
27   \vspace{1.0em}%
28   \noindent%
29   \textbf{\hypertarget{#1}{\etiqueta}}.\par%
30   \nopagebreak[4]%
31   \begin{hangparas}{2em}{0}%
32     \noindent%
33     \textbf{Licencia:} #4\par%
34     \nopagebreak[4]%
35     \noindent%
36     \textbf{Página oficial:} \url{#5}\par%
37     \nopagebreak[4]%
38     \noindent%
39     #6\par%
40   \end{hangparas}%
41   \pagebreak[3]%
42   % Estado previo
43   \setlength{\parskip}{\separacionAnterior}%
44   \renewcommand{\etiqueta}{}}
```

Código fuente 51: Comando para las dependencias.

DOCUMENTOS_ENTREGABLES/REPORTE_TECNICO/PLANTILLAS/DEPENDENCIA.STY

```
64 diagramas se encuentran dentro del seguimiento del control de versiones; con
65 un poco de cuidado, se puede mantener separada a la forma del contenido, lo
66 cuál permite crear diagramas con exactamente el mismo aspecto. Otra ventaja
67 es la integración entre los diagramas y el propio documento: Ti\textit{k}Z
68 produce comandos en \gls{gl:pgf} que se encuentran incrustados directamente
69 en el \gls{gl:pdf} producto, lo cual implica que el texto de los diagramas es
70 seleccionable (en algunos diagramas incluso hay hipervínculos, por ejemplo,
71 el diagrama de estados"\ref{estados_actores}); la tipografía es
72 la misma para diagramas y documento. La única desventaja está dada por el
73 tiempo de aprendizaje de la sintaxis, que en comparación con un editor
74 gráfico, es algo mucho más lento. A modo de ejemplo, el código
75 fuente"\ref{codigo:diagrama_tikz} muestra un fragmento del archivo fuente del
76 diagrama"\ref{diagrama:aes_add_round_key}.
77
78 \codigoFuente[codigo:diagrama_tikz]{1}{38}{latex}{%
79   documentos_entregables/reporte_tecnico/contenidos/marco_teorico/%
80   bloques/diagramas/addRoundKey.tikz.tex}{%
81   Ejemplo de diagrama en Ti\textit{k}Z;
82   diagrama"\ref{diagrama:aes_add_round_key}}
83
84 \dependencia{Diagramas \acrshort{gl:uml}}{Ti\textit{k}Z-\acrshort{gl:uml}}{%
```

Código fuente 52: Uso del comando para dependencias.

DOCUMENTOS_ENTREGABLES/REPORTE_TECNICO/CONTENIDOS/APENDICES/SOBRE_LATEX/SOBRE_LATEX.TEX

```
1 %
2 % Caso de uso para eliminar el registro de un cliente.
3 % Capítulo de análisis y diseño de api web,
4 % Proyecto Lovelace.
5 %
6
7 \casoDeUso[cu:eliminar_cliente]
8 {Eliminar cuenta de un cliente}
9 {
10    Permite a un usuario tipo \textbf{cliente} eliminar su cuenta.
11
12 \begin{trayectoriaPrincipal}
13
14     \item El cliente presiona \textit{eliminar cuenta} en la
15         interfaz \hipervinculo{iu:control}.
16
17     \item El sistema muestra el mensaje \hipervinculo{msj:adv_eliminar_cliente}.
18
19     \item El cliente presiona el botón \textit{aceptar};
20         [\hipervinculoLocal{ta:cancelar}].
21
22     \item El sistema elimina el registro del cliente en la base de datos
23         (\hipervinculo{rn:estados_correo}).
24
25     \item El sistema elimina todos los tokens (\hipervinculo{rn:estados_token})
26         y las llaves (\hipervinculo{rn:estados_llave}) relacionadas con el
27         cliente.
28
29     \item El sistema elimina las variables del sesión del cliente.
30
31     \item El sistema muestra la interfaz \hipervinculo{iu:inicio}.
32
33 \end{trayectoriaPrincipal}
34
35 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
36
37 \begin{trayectoriaAlternativa}[ta:cancelar]
38     {El cliente cancela la operación}
39
40     \item El cliente presiona el botón \textit{cancelar}.
41
42     \item El sistema muestra la interfaz
43         \hipervinculo{iu:control}.
44
45 \end{trayectoriaAlternativa}
46 }
```

Código fuente 53: Ejemplo de definición de caso de uso.

DOCUMENTOS_ENTREGABLES/REPORTE_TECNICO/CONTENIDOS/ANALISIS_Y_DISENIO_API_WEB/ANALISIS/
CASOS_DE_USO/ELIMINAR_CLIENTE.TEX

Este reporte técnico se terminó de imprimir en el mes de noviembre de 2018 con un tiraje de 10 en la Ciudad de México, México.

