Estudio y comparación de métodos de tokenización

Daniel Ayala Zamorano, Laura Natalia Borbolla Palacios, Ricardo Quezada Figueroa

Escuela Superior de Cómputo, Instituto Politécnico Nacional
daz23ayala@gmail.com, laura@ejemplo.com, qf7.ricardo@gmail.com

Resumen La tokenización consiste en el reemplazo de información sensible por valores sustitutos, llamados tokens, en donde el camino de regreso, del token a la información sensible, no es factible. En los últimos años este proceso se ha vuelto muy popular entre los comercios en línea, pues les permite descargar parte de las responsabilidades de seguridad adquiridas al manejar números de tarjetas de crédito en un tercero, proveedor de servicios de tokenización. Lamentablemente, existe una gran cantidad de desinformación alrededor de cómo generar los tokens, principalmente producida por las estrategias publicitarias de las empresas tokenizadoras, en donde cada una intenta convencer al comprador de que su sistema es el mejor, sin explicar realmente qué es lo que hacen para generar tokens. Uno de los mensajes más comunes entre la publicidad es que la criptografía y la tokenización son cosas distintas, y la segunda es mucho más segura. En este trabajo se explica a detalle en qué consiste la tokenización y cuál es su relación con la criptografía; se revisan y comparan los desempeños de los métodos más comunes para tokenizar; para terminar se concluye con una discusión alrededor de las ventajas y desventajas de cada uno.

1. Introducción

Cuando el comercio a través de Internet comenzó a popularizarse, los fraudes de tarjetas bancarias se volvieron un problema alarmante: Visa y MasterCard reportaron, entre 1988 y 1998, pérdidas de 750 millones de dólares; según [12], en el año 2001 se tuvieron pérdidas de 1.7 miles de millones de dólares y, para 2002 aumentaron a 2.1. Como una medida para proteger los sistemas procesadores de pagos, las principales compañías de tarjetas de crédito publicaron un estándar obligatorio para todos aquellos que procesaran más de 20 000 transacciones anuales: el PCI DSS (en inglés, *Payment Card Industry Data Security Standard* [8]).

El PCI DSS cuenta con una gran cantidad de requerimientos, entre ellos, controles estrictos de acceso, monitoreo regular de redes, mantener programas de vulnerabilidades y políticas de seguridad de información,

etcétera [2] [14]; por lo que, para negocios medianos y pequeños, resulta muy difícil obtener un resultado positivo. Además, a pesar de la publicación del estándar en 2004, han seguido existiendo grandes filtraciones de datos (*TJX* en 2006, *Hannaford Bros.* en 2008, *Target* en 2013, *Home Depot* en 2014, por mencionar algnos ejemplos).

Es en este contexto en el que surge el enfoque de la tokenización. Hasta antes de este momento, la idea era proteger la información sensible en donde quiera que se encontrase. Si los números de tarjetas de los usuarios se encontraban regados en diversas pates de un sistema, había que proteger todas esas partes. La tokenización consiste en concentrar la información sensible en un solo lugar para hacer la tarea de protección más sencilla. Al momento de ingreso de un nuevo valor sensible, por ejemplo, la información bancaria de un usuario, se genera un token ligado a esa información: el token se usa en todo el sistema y la información sensible se protege en un solo lugar. Un posible adversario con acceso a los tokens no debe poder obtener la información sensible a partir de estos.

Una de las ventajas de la tokenización es que puede verse como un sistema autónomo, independiente del sistema principal. De esta manera se establece una separación de responsabilidades: el sistema principal se ocupa de la operación del negocio (por ejemplo, una tienda en línea) y el sistema tokenizador se dedica a la protección de la información sensible. Hoy en día varias compañías ofrecen serivicios de tokenización que permiten que los comerciantes se libren casi por completo de cumplir con el PCI DSS. En la figura 1 se muestra una distribución bastante común para un comercio en línea: el sistema tokenizador guarda la información sensible en su base de datos y se encarga de realizar las transacciones bancarias.

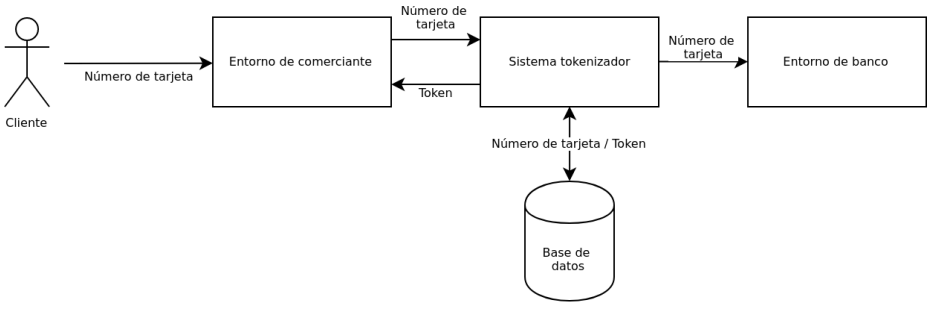

Figura 1. Arquitectura típica de un sistema tokenizador.

2. Preliminares

2.1. Notación

Se denotarán a todas las cadenas de bits de longitud n como $\{0,1\}^n$.

2.2. Estructura de un número de tarjeta bancaria

También llamado PAN por sus siglas en inglés, se refiere al número de una tarjeta bancaria, está compuesto por tres partes:

1. IIN
2. Número de cuenta
3. Dígito verificador

La longitud del número de tarjeta puede variar entre 12 y 19 dígitos y el primero conjunto de números está regido bajo el estándar ISO/IEC-7812.

El IIN (*Número de identificación del emisor* por sus siglas en inglés), está compuesto por los primeros seis dígitos de la tarjeta; permite identificar el banco emisor, el tipo de la tarjeta, la marca (Visa, AmericanExpress) y el nivel de la tarjeta (Clásica, Gold). El primer dígito del IIN es conocido como MII (*Identificador principal de la Industria* por sus siglas en inglés) y su función es señalar la rama de la industria a la que pertenece la entidad que emitió la tarjeta; por ejemplo, los bancos y la industria financiera tienen asignados los números 4 y 5 [11].

Los dígitos que le siguen al INN, excepto el último, son los que componen el número de cuenta y su tamaño varía dependiendo de la longitud del PAN; la longitud máxima, sin embargo, es de 12 dígitos, por lo que cada emisor tiene 10^{12} posibles números de cuenta.

El dígito verificador es calculado mediante el algoritmo de Luhn y su propósito es ayudar a distinguir entre un PAN válido y un PAN inválido. El algoritmo se describe a continuación.

2.3. Algoritmo de Luhn

La especificación de este algoritmo se encuentra en [11]. Se tiene como entrada un número de tarjeta $x = \{x_n, x_{n-1}, \ldots, x_2, x_1\}$ de longitud n; para calcular el dígito verificador se hace lo siguiente:

1. Obtener los conjuntos $x_{par} = \{x_2, x_4, \ldots\}$ y $x_{impar} = \{x_3, x_5, \ldots\}$.

2. Obtener el doble de cada uno de los elementos del conjunto x_{par}. $x_{par\_doble} = \{2 \times x_2, 2 \times x_4, \dots\}$. $\forall x_i \in x_{par\_doble}(x_i > 9 \rightarrow x_i = (x_i \text{ mód } 10) + 1)$.
3. Obtener la suma S de los elementos de los conjuntos x_{par_doble} y x_{impar}.
4. Finalmente, $x_1 = (S \times 9) \text{ mód } 10$

2.4. Cifrado por bloques

Un cifrado por bloques es un cifrado simétrico que se define por la función $E : \mathcal{M} \times \mathcal{K} \rightarrow \mathcal{C}$ en donde $\mathcal{M}$ es el espacio de textos en claro, $\mathcal{K}$ es el espacio de llaves y $\mathcal{C}$ es el espacio de mensajes cifrados. Tanto los mensajes en claro como los cifrados tienen una misma longitud n, que representa el tamaño del bloque [13].

Los cifrados por bloque son un elemento de construcción fundamental para otras primitivas criptográficas. Muchos de los algoritmos tokenizadores que se presentan en este trabajo los ocupan de alguna forma. Las definiciones de los algoritmos son flexibles en el sentido de que permiten instanciar cada implementación con el cifrado por bloques que se quiera; en el caso de las implementaciones hechas para este trabajo se ocupó AES (*Advanced Encryption Standard*) en la mayoría de los casos.

2.5. Cifrado que preserva el formato

Un cifrado que preserva el formato (en inglés *Format-preserving Encryption*, FPE) puede ser visto como un cifrado simétrico en donde el mensaje en claro y el mensaje cifrado mantienen un formato en común. Formalmente, de acuerdo a lo definido en [4], se trata de una función $E : \mathcal{K} \times \mathcal{N} \times \mathcal{T} \times \mathcal{X} \rightarrow \mathcal{X}$, en donde los conjuntos $\mathcal{K}$, $\mathcal{N}$, $\mathcal{T}$, $\mathcal{X}$ corresponden al espacio de llaves, espacio de formatos, espacio de *tweaks* y el dominio, respectivamente. El proceso de cifrado de un elemento del dominio con respecto a una llave K, un formato N y un *tweak* T se escribe como $E_K^{N,T}(X)$. El proceso inverso es también una función $D : \mathcal{K} \times \mathcal{N} \times \mathcal{T} \times \mathcal{X} \rightarrow \mathcal{X}$, en donde $D_K^{N,T}\left(E_K^{N,T}(X)\right) = X$.

Para lo que a este trabajo respecta, el formato usado es el de las tarjetas de crédito: una cadena de entre 12 y 19 dígitos decimales. Esto es $N = \{0, 1, \dots, 9\}^n$ en donde $12 \leq n \leq 19$.

En marzo de 2016 el NIST (*National Institute of Standards and Technology*) publicó un estándar referente a los cifrados que preservan el formato[10]. En él se definen dos posibles métodos: FF1 (lo que en este trabajo es FFX) y FF3 (lo que en este trabajo es BPS).

2.6. Generadores de números pseudoaleatorios

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. Este tipo de generadores se conocen como *no determinísticos* (NRBG, *Non-deterministic Random Bit Generator*) o *realmente aleatorios* (TRBG, *Truly Random Bit Generator*). La segunda manera, que es la que se explicará a detalle en esta sección, es calcular los bits de forma determinística mediante un algoritmo. Este tipo de generadores se conocen como *determinísticos* (DRBG, *Deterministic Random Bit Generator*), y por lo tanto, los números que genera se denominan *pseudoaleatorios*.

En [3] el NIST establece un estándar para dos generadores pseudoaleatorios: uno basado en funciones hash y el otro en cifrados por bloque. La idea general para ambos es la misma: a partir de un valor inicial, llamado semilla, usar el mecanismo interno (la función hash o el cifrado por bloque) a lo largo de las distintas peticiones sobre el generador para producir cadenas de bits de aspecto aleatorio. Las producciones del generador son impredecibles mientras la semilla se mantenga en secreto. La mejor práctica es que la semilla sea producto de un generador no determinístico.

El método para generar bits pseudoaleatorios con una función hash consiste en ir concatenando de forma consecutiva los valores hash derivados de la semilla hasta alcanzar el número de bytes deseados. Primero se genera un hash de la semilla y después se incrementa su valor; de esta forma nunca se obtiene el mismo hash dos veces. Por otra parte, el método basado en un cifrado por bloques consiste en usar el modo de operación de contador con un cifrado por bloques estándar (AES o TDES), en donde la semilla juega el papel del vector de inicialización.

3. Algoritmos tokenizadores

Como el enfoque de este artículo es ver a la tokenización como un servicio (figura 1), la interfaz para los procesos de tokenización y detokenización, desde el punto de vista de los usuarios del servicio, es sumamente simple: el proceso de tokenización es una función $E : \mathcal{X} \to \mathcal{Y}$ y el de detokenización es simplemente la función inversa $D : \mathcal{Y} \to \mathcal{X}$, en donde $\mathcal{X}$ y $\mathcal{Y}$ son los espacios de números de tarjetas y tokens, respectivamente. Ambos conjuntos son cadenas de dígitos de entre 12 y 19 caracteres. Los números de tarjeta cuentan con un dígito verificador que hace que `algoritmoDeLuhn(X) = 0`; los tokens cuentan con un dígito verificador que hace que `algoritmoDeLuhn(Y) = 1`. El último punto es con el

propósito de que sea posible distinguir entre un número de tarjeta y un token.

El PCI SSC (*Payment Card Industry Security Standard Council*) establece en sus guías de tokenización la siguiente clasificación para los algoritmos tokenizadores [7]:

- Métodos reversibles. Aquellos para los cuales es posible regresar al número de tarjeta a partir del token.
  - Criptográficos. Ocupan un esquema de cifrado simétrico: el número de tarjeta y una llave entran al mecanismo de tokenización para obtener un token; el token y la misma llave entran al mecanismo de detokenización para obtener el número de tarjeta original.
  - No criptográficos. Ocupan una base de datos para guardar las relaciones entre números de tarjetas y tokens; el proceso de detokenización simplemente es una consulta a a la base de datos.
- Métodos irreversibles. Aquellos en los que no es posible regresar al número de tarjeta original a partir del token.
  - Autenticable. Permiten validar cuando un token dado corresponde a un número de tarjeta dado.
  - No autenticable. No permiten hacer la validación anterior.

La denominación *no criptográficos* resulta totalmente confusa, pues en realidad todos los métodos conocidos que caen en las categorías de arriba ocupan primitivas criptográficas. La segunda categoría (los irreversibles) carece de utilidad para aplicaciones que procesan pagos con tarjetas de crédito, pues la habilidad de regresar al número de tarjeta a partir de su token es uno de los requerimientos principales para los sistemas tokenizadores. Por lo anterior, en este trabajo se propone una clasificación distinta:

- Métodos criptográficos. Todos aquellos que ocupan herramientas critográficas para operar.
  - Reversibles. Ocupan un esquema de cifrado simétrico: el número de tarjeta y una llave entran al mecanismo de tokenización para obtener un token; el token y la misma llave entran al mecanismo de detokenización para obtener el número de tarjeta original. El término *reversible* es porque se puede regresar al número de tarjeta sin ayuda de herramientas externas, como una base de datos.
  - Irreversibles. Ocupan herramientas criptográficas para generar el token de un número de tarjeta. Operan como funciones de un solo sentido: la única manera de regresar al número de tarjeta a partir

de un token es mediante un ataque por fuerza bruta o mediante herramientas externas, como una base de datos.

- Métodos no criptográficos. Aquellos posibles métodos que no ocupen herramientas relacionadas con la criptografía; por ejemplo, un generador de números realmente aleatorio (TRNG, *True Random Number Generator*).

La clasificación de los *no criptográficos* solamente se propone para abarcar métodos de los cuales realmente se pueda decir que no se relacionan con la criptografía. En este trabajo no se presenta ningún método que clasifique en esa categoría.

A continuación se presentan algunos de los algoritmos tokenizadores más comunes. Al final de cada sección se explica en qué categoría cae según las dos clasificaciones anteriores.

3.1. TKR

En [9] se analiza formalmente el problema de la generación de tokens y se propone un algoritmo que no está basado en cifrados que preservan el formato. Hasta antes de la publicación de este documento, los únicos métodos para generar tokens cuya seguridad estaba formalmente demostrada eran los basados en cifrados que preservan el formato.

El algoritmo propuesto usa un cifrado por bloques para generar tokens pseudoaleatorios y almacena en una base de datos la relación original de estos con los números de tarjetas. En la figura 2 se muestra el proceso de tokenización y detokenización.

Las funciones `buscarTarjeta`, `buscarToken` e `insertar` sirven para interactuar con la base de datos. Lo único que queda por esclarecer es el el contenido de la función generadora de tokens pseudoaleatorios, la función `RN`. El algoritmo de esta función se muestra en la figura 3. Idealmente, esta función debe regresar un elemento uniformemente aleatorio del espacio de tokens. 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 $\lambda = \lceil \log_2 l \rceil$.

Existen varios candidatos viables para la función f: un cifrado de flujo, pues el flujo de llave de estos produce cadenas de aspecto aleatorio, o un cifrado por bloques con un modo de operación de contador. En la implementación de este trabajo se ocupa esta última opción.

Algoritmo TKR-tokenización(x, k)
1. $q \leftarrow$ `buscarTarjeta(`x`)`
2. **si** $q = 0$ **entonces**:
3. $\quad y \leftarrow$ `RN(`k`)`
4. $\quad$ `insertar(`x`, `y`)`
5. **sino:**
6. $\quad y \leftarrow q$
7. **regresar** y

Algoritmo TKR-detokenización(y, k)
1. $q \leftarrow$ `buscarToken(`t`)`
2. **si** $q = 0$ **entonces**:
3. $\quad$ **regresar error**
4. **sino:**
5. $\quad$ **regresar** q

Figura 2. Tokenización y detokenización de TKR

Algoritmo TKR-RN(k)
1. $x \leftarrow$ `f(`k`, contador)`
2. $x_1, x_2, \ldots, x_m \leftarrow$ `cortar(`x`, `λ`)`
3. $t \leftarrow$, $i \leftarrow 0$
4. **mientras** $|t| \neq \mu$:
5. $\quad$ **si** `entero(`x_i`)` **entonces**:
6. $\quad\quad t \leftarrow t+$ `entero(`X_i`)`
7. $\quad i \leftarrow i + 1$
8. `contador` $\leftarrow$ `contador` $+1$
9. **regresar** t

Figura 3. Generación de tokens pseudoaleatorios en TKR

Con la clasificación del PCI, este método cae, contradictoriamente, en los reversibles no criptográficos. Con la clasificación propuesta en este trabajo se encuentra dentro de los criptográficos irreversibles.

3.2. FFX (*Format-preserving Feistel-based Encryption*)

Cifrado que preserva el formato presentado en [5] 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. De aquí en adelante se hablará solamente de la colección A10.

FFX ocupa una red Feistel alternante junto con una adaptación de AES-CBC-MAC (usada como función de ronda) para lograr preservar el formato. La operación general del algoritmo se describe completamente por la operación de una red alternante:

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

En la figura 4 se describe a la función de ronda. La idea general consiste en interpretar la salida de AES CBC MAC de forma que tenga el formato deseado. El valor de m corresponde al *split* en la ronda actual, esto es, la longitud de la cadena de entrada.

Con la clasificación del PCI, este método cae en los reversibles criptográficos. Con la clasificación propuesta en este trabajo, se trata de un criptográfico reversible.

3.3. BPS

Algoritmo de cifrado que preserva el formato capaz de cifrar cadenas formadas por cualquier conjunto de caracteres, descrito en [6] y cuyo nombre proviene de las iniciales de los apellidos de sus autores Eric Brier, Thomas Peyrin y Jacques Stern, aunque en el estándar [10], el NIST lo nombra como FF3.

BPS se conforma de 2 partes: un cifrado interno BC que se encarga de cifrar bloques de longitud fija, usando a su vez un cifrado por bloques F; y

Algoritmo FFX-AES-CBC-MAC(x, k, t)
1. $a \leftarrow x \mid\mid t$
2. $b \leftarrow$ `aes_cbc_mac(`a`,` k`)`
3. $y\prime \leftarrow$ a[1 ... 64]
4. $y\prime\prime \leftarrow$ a[65 ... 128]
5. **si** m ≤ 9 **entonces**:
6. c $\leftarrow y\prime\prime \mod 10^m$
7. **sino**:
8. c $\leftarrow (y\prime \mod 10^{m-9}) \times 10^9 + (y\prime\prime \mod 10^m)$
9. **regresar** c

Figura 4. Función de ronda de FFX A10.

un modo de operación especial, encargado de extender la funcionalidad de BC y permitir cifrar cadenas de un longitud de hasta $max_b \cdot 2^{16}$ caracteres, donde max_b es la longitud máxima que puede tener una cadena para cifrarse con BC.

Este cifrado interno utiliza una red Feistel alternante y se define como $BC_{F,s,b,w}(X, K, T)$, donde: F es un cifrado por bloques con f bits de salida, como puede ser TDES o AES; s es la cardinalidad del alfabeto de la cadena a cifrar, b es su longitud, w es el número de rondas de la red Feistel, X es la cadena, K es una llave acorde al cifrado F, y T es un *tweak* de 64 bits.

El funcionamiento del cifrado BC es descrito en la figura 5.

Para cada bloque a cifrar, el cifrado BC debe instanciarse con una longitud de $max_b = 2 \cdot log_s(2^{f-32})$ caracteres, y cuando la longitud total del mensaje a cifrar no sea múltiplo de este valor, en el último bloque BC se tendrá que instanciar con una longitud igual a la de ese bloque.

El modo de operación de BPS es un variación de CBC, con la diferencia de que usa sumas modulares carácter por carácter en lugar de aplicar operaciones *xor*, además de que no emplea un vector de inicialización.

Otra característica de este modo de operación es que utiliza un contador u para aplicar un *xor* a los 16 bits más significativos de cada mitad del *tweak* T que utiliza BPS, por lo cual este se puede ver como una función $u(n) = n \cdot (2^{16} + 2^{48})$.

El funcionamiento del modo de operación se describe en la figura 6.

El PCI clasifica a este algoritmo dentro de los método de de generación de tokens reversibles criptográficos, pero desde el punto de vista de este trabajo se tiene que es un método criptográfico reversible.

Algoritmo Cifrado $BC_{F,s,b,w}(X,K,T)$

1. $T_R \leftarrow T \mod 2^{32}$ y $T_L \leftarrow (T - T_R)/2^{32}$
2. $l \leftarrow \lceil b/2 \rceil$
3. $r \leftarrow \lfloor b/2 \rfloor$
4. $L_0 \leftarrow \sum_{j=0}^{l-1} X[j] \cdot s^j$
5. $R_0 \leftarrow \sum_{j=0}^{r-1} X[j+l] \cdot s^j$
6. **para** $i = 0$ **hasta** $i = w - 1$:
7. **si** i es par:
8. $L_{i+1} \leftarrow L_i \boxplus F_K((T_R \oplus i) \cdot 2^{f-32} + R_i) \pmod{s^l}$
9. $R_{i+1} \leftarrow R_i$
10. **si** i es impar:
11. $R_{i+1} \leftarrow R_i \boxplus F_K((T_L \oplus i) \cdot 2^{f-32} + L_i) \pmod{s^r}$
12. $L_{i+1} \leftarrow L_i$
13. **para** $i = 0$ **hasta** $i = l - 1$:
14. $Y_L[i] \leftarrow L_w \bmod s$
15. $L_w \leftarrow (L_w - Y_L[i])/s$
16. **para** $i = l$ **hasta** $i = r - 1$:
17. $Y_R[i] \leftarrow R_w \bmod s$
18. $R_w \leftarrow (R_w - Y_R[i])/s$
19. $Y \leftarrow Y_L \parallel Y_R$

Figura 5. Cifrado interno BC.

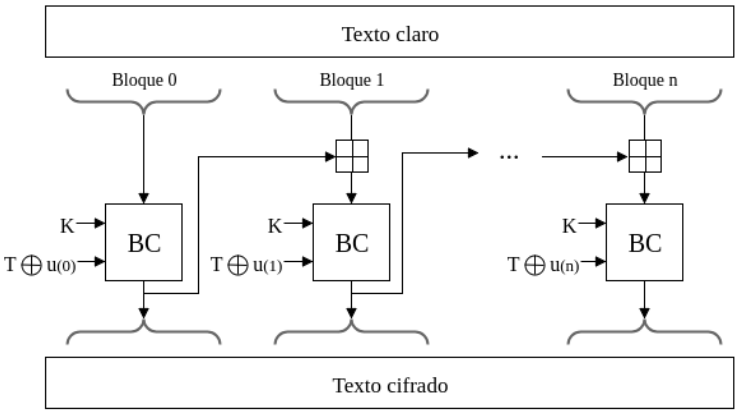

Figura 6. Modo de operación de BPS.

3.4. Algoritmo basado en generador pseudoaleatorio

Probablemente este método es el más directo para generar tokens. La idea es producir una cadena binaria aleatoria con un DRBG e interpretarla para que tenga el formato de un token. El funcionamiento general es el mismo que en TKR (figura 2): la operación de tokenización primero verifica en la base de datos que el número de tarjeta no se encuentre ya asociado a un token; de ser este el caso, se regresa el token asociado previamente; en caso contrario se genera un nuevo token aleatorio, se guarda en la base de datos y se regresa. La detokenización es simplemente una consulta en la base de datos. En la figura 7 se muestra uno de los posibles métodos para generar un token a partir de una cadena binaria (n es la longitud que debe tener el token generado).

Algoritmo DRBG-tokenización(n)
1. `token` ←
2. `cadena_aleatoria` ← `drbg.generar(`n`)`
3. **para** $i = 0$ **hasta** $n - 1$:
4. `token[`i`]` ← `cadena_aleatoria[`i`]` mód 10
5. **regresar** token

Figura 7. Generación de tokens a partir de cadena binaria aleatoria.

3.5. AHR (Algoritmo híbrido reversible

En 2017, Longo, Aragona y Sala [1] propusieron un algoritmo híbrido reversible basado en un cifrador por bloques, una llave secreta y una entrada adicional. Las entradas del algoritmo son la parte del PAN a cifrar y una entrada adicional (por ejemplo, la fecha) que permite que se tengan varios tokens relacionados con la misma tarjeta.

El algoritmo necesita una función f pública que se encarga de poner el relleno en la entrada para obtener el bloque completo para el cifrador, pues, dada una cadena de longitud m regrese una de longitud n; requiere también que solo se utilicen cifrados cuyo tamaño de bloque sea, mínimo, de 128 bits. Finalmente, como es un algoritmo reversible, se necesita una base de datos segura para almacenar los pares PAN-token.

Como se desea obtener un token que tenga el mismo número de dígitos que el PAN ingresado, se utiliza un método llamado *cycle-walking* para asegurarse de que el texto cifrado pertenezca el espacio del texto en claro.

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 primer paso es obtener el valor del bloque t; es decir, concatenar los bytes más significativos de la salida de $f(u, p)$ con la representación binaria de p. Después, se cifra el bloque t con la llave K y se guarda en c la representación decimal de los últimos bits del bloque cifrado; aquí es donde se utiliza la caminata cíclica, pues si los dígitos de c son menores a los que le corresponden con p, se guarda t en c y se regresa al paso del cifrado. Finalmente, cuando se obtiene un token válido, se comprueba que no esté registrado en la base de datos (si lo está, se regresa al inicio, pero aumentando la entrada adicional u en 1) y se registra el nuevo par PAN-token. El pseudocódigo del algoritmo de tokenización se puede observar en 8.

Algoritmo AHR(p, u, k)
1. $t = f(u, p)||[\bar{p}]_b^s$
2. $c = E(k, t)$
3. **si** ($\bar{c}$ mód 2^n) $\geq 10^l$ **entonces**:
4. $t = c$
5. Regresar a 2.
5. $token = [\bar{c}$ mód $2^n]_{10}^l$
5. **si** $comprobar(token)$ = verdadero **entonces**:
6. $u = u + 1$
6. Regresar al paso 1.
7. **sino**:
9. **regresar** $token$

Figura 8. Algoritmo híbrido reversible.

4. Resultados de comparaciones de desempeño

Todos los resultados presentados en esta sección se llevaron a cabo en una computadora con las siguientes características:

Procesador: Intel i5-7200U (2.5 GHz) de 4 núcleos.
Sistema operativo: Arch Linux, kernel 4.17.
Base de datos: MariaDB 10.1.
Compilador: GCC 8.1.1

En la tabla 1 y la figura 9 se muestran los resultados en tiempo de las ejecuciones de los algoritmos presentados en secciones anteriores.

Tabla 1. Comparación de tiempos de tokenización.

Algoritmo	Tokenización (μs)	Detokenización (μs)
FFX	78	60
BPS	331	181
TKR	58773	717
AHR	4584	1201
DRBG	54473	391

5. Conclusiones

Referencias

1. R. Aragona, R. Longo, and M. Sala. Several proofs of security for a tokenization algorithm. *Appl. Algebra Eng. Commun. Comput.*, 28(5):425–436, 2017.
2. U. C. Association. What is pci dss?, 2018.
3. E. Barker and J. Kelsey. Nist special publication 800-90a - recommendation for random number generation using deterministic random bit generators, 2015.
4. M. Bellare, T. Ristenpart, P. Rogaway, and T. Stegers. Format-preserving encryption. In M. J. J. Jr., V. Rijmen, and R. Safavi-Naini, editors, *Selected Areas in Cryptography, 16th Annual International Workshop, SAC 2009, Calgary, Alberta, Canada, August 13-14, 2009, Revised Selected Papers*, volume 5867 of *Lecture Notes in Computer Science*, pages 295–312. Springer, 2009.
5. M. Bellare, P. Rogaway, and T. Spies. The ffx mode of operation for format-preserving encryption. 2009.
6. E. Brier, T. Peyrin, and J. Stern. Bps: a format-preserving encryption proposal. 2010.
7. P. C. I. S. S. Council. Tokenization product security guidelines – irreversible and reversible tokens, 2015.

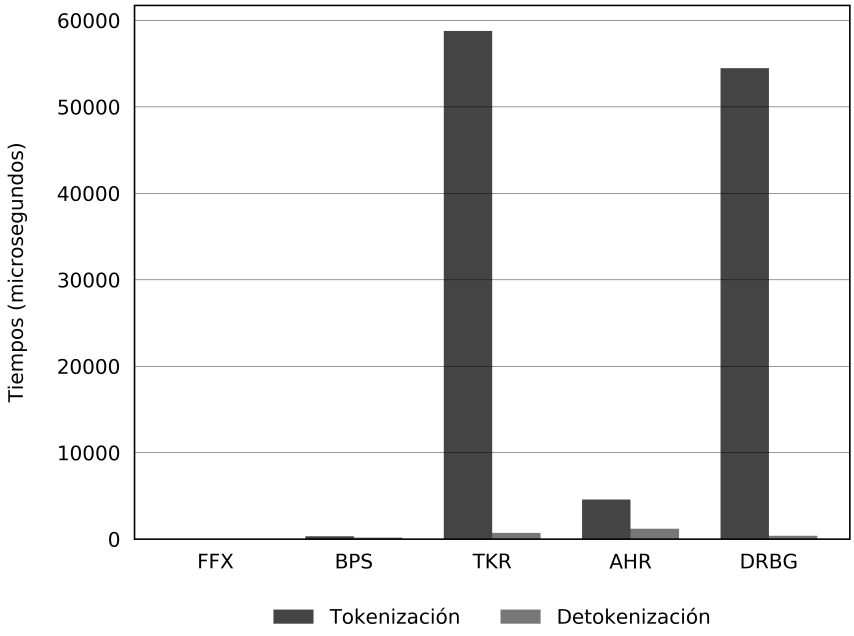

Figura 9. Comparación de tiempos de tokenización.

8. P. C. I. S. S. Council. Data security standard - version 3.2, 2016.
9. S. Diaz-Santiago, L. M. Rodríguez-Henríquez, and D. Chakraborty. A cryptographic study of tokenization systems. *Int. J. Inf. Sec.*, 15(4):413–432, 2016.
10. M. Dworkin. Nist special publication 800-38g - recommendation for block cipher modes of operation: Methods for format-preserving encryption, 2016.
11. I. O. for Standarization. *ISO/IEC 7812*. 5 edition, 2017.
12. J. S. Kiernan. Credit card and debit card fraud statistics, 2017.
13. A. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.
14. S. Staff. The history of the pci dss standard: A visual timeline, 2013.