AAECC (2017) 28:425–436
DOI 10.1007/s00200-017-0313-3

ORIGINAL PAPER

Several proofs of security for a tokenization algorithm

Riccardo Aragona[1] · **Riccardo Longo**[1] ·
Massimiliano Sala[1]

Received: 12 October 2016 / Accepted: 1 February 2017 / Published online: 13 February 2017

Abstract In this paper we propose a tokenization algorithm of Reversible Hybrid type, as defined in PCI DSS guidelines for designing a tokenization solution, based on a block cipher with a secret key and (possibly public) additional input. We provide some formal proofs of security for it, which imply our algorithm satisfies the most significant security requirements described in PCI DSS tokenization guidelines. Finally, we give an instantiation with concrete cryptographic primitives and fixed length of the PAN, and we analyze its efficiency and security.

Keywords Tokenization · Block ciphers · Provable security · IND CPA security

Mathematics Subject Classification 94A60

1 Introduction

In recent years, credit cards have become one of the most popular payment instruments. Their growing popularity has brought many companies to store the card information of its customers to make simpler subsequent payments. This need is shared by many other actors in the payment process. On the other hand, credit card data are very sensitive information, and then the theft of such data is considered a serious threat.

Any company that stores credit card data aims to achieve the *Payment Card Industry Security Standard Council* (*PCI SSC*) compliance. The PCI SSC is an organization, founded by the largest payment card networks, which has developed several standards and recommendations. One of these is called the *PCI Data Security Standard* (*PCI*

✉ Riccardo Aragona
ric.aragona@gmail.com

[1] University of Trento, Trento, Italy

DSS [14]) and its goal is to guarantee the security of credit card data. PCI DSS requires that companies that handle payment cards protect the data of the cardholder when these are stored, transmitted or processed.

These stringent requirements led to consider a new method for storage and transmission of the card information: instead of protecting the actual card data, it is easier to remove them (when their storing is not requested) and replace them with another value, called *token*. Tokens are alpha-numeric strings representing the *PAN* (*Primary Account Number*) of a payment card, and that may have a format similar to it. In any case, from a token it must be infeasible (without additional information) to recover the PAN from which it was generated. This process is called *tokenization*. In [5], the authors present an interesting formal cryptographic study of tokenization systems and their security.

In recent years, PCI SSC has drafted some guidelines to design a tokenization solution [12,13]. In [13], the following five types of tokens are described: *Authenticatable Irreversible Tokens*, *Non-Authenticatable Irreversible Tokens*, *Reversible Cryptographic Tokens*, *Reversible Non-Cryptographic Tokens* and *Reversible Hybrid Tokens*.

In particular, a reversible tokenization algorithm , i.e. providing the possibility for entities using or producing tokens to obtain the original PAN from the token, can be designed in three ways:

- *Reversible Cryptographic*, if it generates tokens from PANs using strong cryptography. In particular, a mathematical relationship between PAN and corresponding token exists. In this case, the PAN is never stored; only the cryptographic key is (securely) stored.
- *Reversible Non-Cryptographic*, if obtaining a PAN from its token is only by a data look-up in a dedicated server (called a Card Data Vault). In this case, the token has no mathematical relationship with its associated PAN and the only thing to be kept secret is the actual relationship between the PAN and its token (e.g., a look-up table in the Card Data Vault).
- A reversible tokenization algorithm is called *Hybrid* if it contains some features of both Reversible Cryptographic tokens and Reversible Non-Cryptographic tokens. A typical situation of this type is when, although there is a mathematical relationship between a token and its associated PAN, a data look-up table must be used to retrieve the PAN from the token.

In this paper we propose a tokenization algorithm of the *Reversible Hybrid* type, based on a block cipher with a secret key and (possibly public) additional input. We provide some formal proofs of security for it. To fully appreciate our design and the proposed proofs it is necessary to analyze the PCI requirements in more detail. The remainder of the paper is thus as follows:

- In Sect. 2, we analyze some PCI requirements for a tokenization algorithm ([12, 13]).
- In Sect. 3, we describe our tokenization algorithm;
- In Sect. 4, we prove the security of the algorithm defined in Sect. 3 in a very general scenario, which would imply our algorithm satisfies most requirements present in Sect. 2. More precisely, we present a security notion, Indistinguishability under a

Chosen-Plaintext Attack (IND-CPA), for a tokenization algorithm of our type, and we prove it under the assumption that our core cryptographic algorithm satisfies a standard IND-CPA. We also provide a separate proof for a special requirement.
– In Sect. 5, we give an instantiation of our algorithm, considering concrete cryptographic primitives and fixing PAN length, and analyzing the security and efficiency in this real-life application.

2 Requirements

The first two requirements are not linked to security:

– Although tokens can enjoy a variety of formats, the most convenient is probably the same format of the PAN itself, since in this case a token can move inside a payment network and also be used as a payment token (for a definition of a *payment token* see p. 13 in [6]). But if we try to create a token by a direct encryption of the PAN we will meet several problems, since the output of the encryption may not have a format like a PAN. Indeed, we must keep in mind that PCI requests the use of standard encryption algorithms and so we cannot create ad-hoc cryptographic primitives, but we must rely on established algorithms.
So we must face the problem of designing algorithms that preserve the message format, or the so-called *Format Preserving Encryption* (*FPE*) [1]. In literature, there are some interesting examples of algorithms that solve such problem [2,4,7, 9,15].
– It must be possible to obtain different tokens from a single PAN (even one per transaction, if necessary), so the tokenization algorithm will require additional inputs, such as, a transaction counter, an expiration date, etc.

Concerning security issues, there are many security requirements that our algorithm has to satisfy in order to meet PCI compliance. We list here the main requests:

A1 "*the recovery of the original PAN must not be computationally feasible knowing only the token or a number of tokens.*" (p. 6 in [12]).
In other words, even if an attacker has managed to collect many tokens, all coming from the same PAN, possibly even on a long period of time, they must be computationally unable to retrieve the corresponding PAN. This is a form of ciphertext-only attack.

A2 "*access to multiple token-to-PAN pairs should not allow the ability to predict or determine other PAN values from knowledge of only tokens.*" (p. 6 in [12] and GT4 in [13]).
This is a known-plaintext attack.

A3 "*Tokens should have no value if compromised or stolen, and should be unusable to an attacker if a system storing only tokens is compromised*" (p. 6 in [12]).
Since this sentence comes immediately after A1 and A2, which aim at preventing PAN recoveries, we take the goal of this rather cryptic sentence to be the prevention of unauthorized token generation. In other words, an attacker possessing many tokens (but not knowing the corresponding PAN's) must be unable to generate even *one* other valid token. This condition is drastically different from

A1, because here we do not require the attacker to be able to deduce any of the involved PAN's. However, there are two matters. First, since they needs to compute *valid* tokens, they must have control on any other input of the tokenization algorithm (such as, a transaction counter). Second, they needs to know for which PAN they can generate a token.

A4 "*Converting from a token produced under one cryptographic key to a token produced under another cryptographic key should require an intermediate PAN state—i.e. invocation of de-tokenization.*" (GT 11 in [13]).
Since our system uses a block cipher with additional (public) input, we take this to mean that if an attacker gets a token obtained by a PAN, a secret key and an additional input, then they must be unable to compute any token corresponding to the same PAN (and same additional input) but to a different key.

A* "*The recovery of the original PAN should be computationally infeasible knowing only the token, a number of tokens, or a number of PAN/token pairs*" (GT 5 in [13]).
This is a repetition of A1 and A2.

To prove that our tokenization algorithm satisfies A1, A2 and A3, we will prove in Sect. 4 that it satisfies an even stronger condition, under the assumption of the strength of the core primitive we are using to define it (a block cipher). Requirement A4 requires a separate proof in Sect. 4.

3 Algorithm

A card number is formed by three concatenated parts: the IIN (also called BIN), that identifies the card Issuer, a numeric code, that identifies the account, and a check digit. We assume to replace the IIN with another fixed code (called a "token BIN" in [6], p. 14), which marks the resulting card number as a token, so we will ignore the first part in the description of our algorithm. We will also compute the check digit as appropriate, so we can discard it too in the forthcoming formal description of our algorithm.

We assume to be able to invoke the encryption function of a block cipher E, just by sending a plaintext and obtaining a ciphertext, with a key that is kept somewhere protected and that we do not need to know. We can think of E as the first *ingredient* of our algorithm. $\mathbb{K}$ denotes the keyspace of E and $K \in \mathbb{K}$ will be any key, so that we can view E as a function $E : \mathbb{K} \times (\mathbb{F}_2)^m \longrightarrow (\mathbb{F}_2)^m$ for some $m \in \mathbb{N}$. With K fixed, E is a permutation acting on the set $(\mathbb{F}_2)^m$ of the m-bit strings. With standard block ciphers we have $m = 64$ or $m = 128$. We assume as usual that the set of its encryption functions forms a random sample of the set of permutations acting on $(\mathbb{F}_2)^m$.

For strings we use a notation like $|0110|_2$, where the index 2 denotes that we are using only symbols from $\{0, 1\}$, i.e. remainders of division by 2.

Our algorithm processes two inputs: a numeric code coming from the PAN and an additional input.

- By *numeric code* we mean a string of ℓ decimal digits, ℓ any agreed number $\ell \in \mathbb{N}$, and we formally define the set of our numeric codes as $\mathbb{P} := \{0, 1, \ldots, 9\}^\ell$. At present, $13 \leq \ell \leq 19$ for numeric codes coming from PANs [6], but we do not need to impose any limitation. We observe that $\mathbb{P}$ is in obvious bijection with the

integer set $\{a \in \mathbb{N} \mid 0 \leq a < 10^{\ell}\}$, and so we can view a numeric code also as a non-negative integer, but care has to be taken to pass from one representation to the other. Let $[y]_b^s$ denote the representation (string) of a positive integer $y < b^s$ in base b with s digits, where the most significant digits are on the *left*.
For example, $[12]_{10}^2 = |12|_{10}$, $[12]_{10}^3 = |012|_{10}$ and $[13]_2^5 = |01101|_2$.
Then any positive integer X such that $X < 10^{\ell}$ can be easily converted to $[X]_{10}^{\ell} \in \mathbb{P}$. We will use a bar to denote the conversion from a string to a number, like $\overline{|12|_{10}} = 12$ and $\overline{|01100|_2} = 12$.

- The role of the additional input is to allow for different tokens corresponding to the same PAN and so it can be anything, such as a transaction counter or an integer denoting the current time. Formally, we will identify it as a binary string of finite but arbitrary length (as for example the binarization of a transaction counter). We will call $\mathbb{U}$ the set containing all these strings, so that any $u \in \mathbb{U}$ is implicitly meant to be an additional input.

Let $n := \lceil \log_2(10^{\ell}) \rceil$ be the maximum number of bits required to represent a number with ℓ decimal digits. Since most of the block ciphers used in real life applications have a block-size of at least 64 bits, and for the maximum length of a PAN $\ell = 19$ we have $n = 64$, we assume that ℓ is such that $n \leq m$.

Now we can present f, the second *ingredient* of our algorithm, which is a public function

$$f : \mathbb{U} \times \mathbb{P} \longrightarrow (\mathbb{F}_2)^{m-n}$$

In other words, given an additional input $u \in \mathbb{U}$ and a numeric code $X \in \mathbb{P}$ coming from the PAN , f returns a string of $m - n$ bits.

We require f to be *collision-resistant*, that is, it must be computationally infeasible to obtain two distinct pairs (X_1, u_1) and (X_2, u_2) such that $f(u_1, X_1) = f(u_2, X_2)$. This requirement compels the image space to have a size large enough to prevent brute-force collision attacks. So in the case of PAN tokenization we need to consider only block ciphers with block size of at least 128 bits, in order to have an image space of dimension at least 64 bits. The purposes of this function f are the followings:

- to pad the input of the tokenization algorithm to match the block size of the cipher;
- to allow the creation of multiple different tokens from the same PAN using the same key, useful for example to change token for each transaction.

The output of f could be seen as a *tweak* in the context of Tweakable Encryption [8]. An example for this f could be a truncated version of a cryptographic hash function.

The third *ingredient* for our tokenization algorithm is a database stored somewhere securely that contains a look-up table of PAN-token pairs. Once we have generated a token, we assume it is inserted in the table. However, to generate a new token, we need to access the database only via a function check that checks if the token is already stored and returns either True or False accordingly.

One of the goals of a tokenization algorithm is to obtain an integer with ℓ decimal digits starting from another integer with the same length. Since $n := \lceil \log_2(10^{\ell}) \rceil < m$, we have to consider only a fraction of the output of E, in particular we will take the n

least significant bits of the output, and then convert this string back to an integer. Given that $10^l < 2^n$, this integer could have $\ell + 1$ decimal digits. To solve this problem we use a method known as *Cycle Walking Cipher* [3], designed to encrypt messages from a space $\mathcal{M}$ using a block cipher that acts on a space $\mathcal{M}' \supset \mathcal{M}$, and obtain ciphertexts that are in $\mathcal{M}$.

We are ready to write down our algorithm.

The *Tokenization Algorithm* $T(K, X, u)$ executes the following steps:

1. $t := f(u, X) \mid\mid \left[\bar{X}\right]_2^n$
2. $c := E(K, t)$
3. if $(\bar{c} \mod 2^n) \geq 10^\ell$, then $t := c$ and go back to step 2
4. $\texttt{token} := [\bar{c} \mod 2^n]_{10}^\ell$
5. if $\texttt{check}(\texttt{token}) = \mathsf{True}$, then $u := u + 1$ and go back to step 1
6. return $\texttt{token}$

The correctness of Algorithm T is obvious, we now discuss its termination.

At Step 3 we check if $(\bar{c} \mod 2^n) \geq 10^\ell$. Since $x \mapsto E(K, x)$ is a random permutation, we expect c to resemble a random binary string in $(\mathbb{F}_2)^m$. Therefore, the number $(\bar{c} \mod 2^n)$ is a random integer in $\{0, \dots, 2^n - 1\}$. Recall that $2^{n-1} < 10^\ell < 2^n$. Therefore, the condition at Step 3 is met with probability $0 < p = \frac{2^n - 10^\ell}{2^n} < 1$. Going back to step 2 another pseudo-random number is computed and the probability that the condition of Step 3 is not satisfied for it is again p. Since the two events can be considered independent, due to the pseudorandomness property of E, the probability of the joint event goes down to p^2, and so on. Therefore, the probability that the algorithm remains stuck at Step 3 is negligible.

At Step 5 we check if we already have the new token in our database . If we have it, we increase u to $u + 1$. We remain stuck only if $f(u, X) = f(u + 1, X)$, but this happens very rarely thanks to the collision resistance of f.

For a more detailed discussion of the probability to meet the conditions at Step 3 and at Step 5, see the instantiation of our algorithm given in Sect. 5.

4 Proof of security

In this section we will prove that the algorithm previously defined is secure in an Indistinguishability under a Chosen-Plaintext Attack scenario, under the condition that its core encrypting algorithm is secure in the same scenario. This will guarantee in particular A1, A2 and A3. Then we will prove A4 separately.

Definition 1 **(IND-CPA)** Let $E(K, m) \to c$ be an encrypting function. An *Indistiguishability under Chosen Plaintext Attack* (IND-CPA) game for E between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ proceeds as follows:

Phase I $\mathcal{A}$ chooses a plaintext m_i and sends it to $\mathcal{C}$, that responds with $c_i = E(K, m_i)$. This phase is repeated a polynomial number of times.

Challenge $\mathcal{A}$ chooses two plaintexts m_0^*, m_1^* (never chosen in Phase I) and sends them to $\mathcal{C}$ that selects $\nu \in \{0, 1\}$ at random and computes $c = E(K, m_\nu^*)$. Then $\mathcal{C}$ sends c to $\mathcal{A}$.

Phase II Phase I is repeated, with the obvious restriction that $\mathcal{A}$ cannot choose m_0^* or m_1^*.

Guess $\mathcal{A}$ guesses $\nu' \in \{0, 1\}$. They wins if $\nu' = \nu$.

We say that the advantage $Adv_{\mathcal{A}}^E$ of $\mathcal{A}$ winning the IND-CPA game for the Encrypting algorithm E is:

$$Adv_{\mathcal{A}}^E = \left| Pr\left[\nu' = \nu\right] - \frac{1}{2} \right|$$

E is said to be *secure in a IND-CPA scenario* if there is no probabilistic polynomial-time algorithm $\mathcal{A}$ that wins the CPA game with more than negligible advantage.

We define a IND-CPA game for T analogously to 1, where messages are replaced by numeric codes and additional inputs, while ciphertexts are replaced by tokens. So the adversary chooses two code/additional input pairs and tries to distinguish which of these corresponds to the token returned by $\mathcal{C}$. The adversary is also able to request other tokens (corresponding to a polynomial number of pairs), that she may choose adaptively.

Definition 2 (IND-CPA for a Tokenization Algorithm) Let $T(K, X, u) \rightarrow \texttt{token}$ be a tokenization algorithm that takes as input a key K, a numeric code $X \in \mathbb{P}$ and an additional input $u \in \mathbb{U}$, and returns a token token. An *Indistiguishability under Chosen Plaintext Attack* (IND-CPA) game for T between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ proceeds as follows:

Phase I $\mathcal{A}$ chooses (X_i, u_i) and sends it to $\mathcal{C}$, that responds with $\texttt{token}_i = T(K, X_i, u_i)$. This phase is repeated a polynomial number of times.

Challenge $\mathcal{A}$ chooses (X_0^*, u_0^*), (X_1^*, u_1^*) (with (X_0^*, u_0^*) and (X_1^*, u_1^*) never chosen in Phase I) and sends them to $\mathcal{C}$ that selects $\nu \in \{0, 1\}$ at random and computes $\texttt{token} = T(K, X_\nu^*, u_\nu^*)$, which sends to $\mathcal{A}$.

Phase II Phase I is repeated, with the obvious restriction that $\mathcal{A}$ cannot choose (X_0^*, u_0^*) or (X_1^*, u_1^*).

Guess $\mathcal{A}$ guesses $\nu' \in \{0, 1\}$, she wins if $\nu' = \nu$.

We say that the advantage $Adv_{\mathcal{A}}^T$ of $\mathcal{A}$ winning the IND-CPA game for the Tokenization Algorithm T is:

$$Adv_{\mathcal{A}}^T = \left| Pr\left[\nu' = \nu\right] - \frac{1}{2} \right|$$

T is said to be secure in a IND-CPA scenario if there is no probabilistic polynomial-time algorithm $\mathcal{A}$ that wins the IND-CPA game with more than negligible advantage.

Theorem 1 *(***IND-CPA Security of Tokenization Algorithm***) Let T be the Tokenization Algorithm described in Sect. 3, and let E be the block cipher used in Step 2 of T. If E is secure in an IND-CPA scenario then T is secure in an IND-CPA scenario.*

Proof Let $\mathcal{C}$ be the challenger in the IND-CPA game for E, and $\mathcal{A}$ be an algorithm that can win the IND-CPA game for T with more than negligible advantage ϵ. We will build a simulator $\mathcal{S}$ that plays the IND-CPA game for E by simulating an IND-CPA game for T and interacting with $\mathcal{A}$.

For Phase I and II we need to show how $\mathcal{S}$ answers to tokenization queries (X_i, u_i) made by $\mathcal{A}$. The function f is publicly known, so $\mathcal{S}$ may compute $t_i := f(u_i, X_i) \mid\mid \left[\overline{X_i}\right]_2^n$ and queries $\mathcal{C}$ for the encryption of t_i, obtaining c_i in response. If $(\bar{c}_i \mod 2^n) \geq 10^l$, then $\mathcal{S}$ queries again $\mathcal{C}$, this time requiring the encryption of c_i, repeating this passage until $\mathcal{C}$ answers with c_i such that $(\bar{c}_i \mod 2^n) < 10^\ell$ (which will eventually happen since $c \mapsto E(K, c)$ is a random permutation). At this point $\mathcal{S}$ answers to the query of $\mathcal{A}$ with $\texttt{token}_i := [\bar{c}_i \mod 2^n]_{10}^l$.

Observe that the c_i's, and hence the $\texttt{token}_i$'s, will be distinct with high probability, since f is collision-resistant, even if the X_i's are identical, as long as the pairs (X_i, u_i)'s are distinct.

In the challenge phase, $\mathcal{S}$ receives from $\mathcal{A}$ two pairs (X_0^*, u_0^*), (X_1^*, u_1^*). $\mathcal{S}$ computes $t_j^* := f(u_j^*, X_j^*) \mid\mid [\overline{X^*}_j]_2^n$ for $j \in \{0, 1\}$ and submits them to $\mathcal{C}$ for the challenge phase of the IND-CPA game for E. $\mathcal{C}$ chooses at random $\nu \in \{0, 1\}$ and will respond with the challenge ciphertext c. If $(\bar{c} \mod 2^n) \geq 10^l$, $\mathcal{S}$ queries $\mathcal{C}$ requiring the encryption of c, repeating this passage until we have $\texttt{token} := (\bar{c} \mod 2^n) < 10^\ell$ (which will eventually happen since $c \mapsto E(K, c)$ is a random permutation). At this point $\mathcal{S}$ sends $\texttt{token}$ to $\mathcal{A}$ as the challenge token of the IND-CPA game for T.

Eventually $\mathcal{A}$ will send to $\mathcal{S}$ its guess ν' for which code has been tokenized into $\texttt{token}$. $\mathcal{S}$ then forwards this guess to $\mathcal{C}$. It is clear that $\mathcal{A}$ guesses correctly if and only if $\mathcal{S}$ guesses correctly since the simulation is seamless.

Note that during the Challenge phase $\mathcal{S}$ is not allowed to send to $\mathcal{C}$ messages submitted during Phase I, and in Phase II $\mathcal{S}$ is not allowed to send to $\mathcal{C}$ the two messages submitted in the Challenge phase. Since the same restriction applies to the interaction between $\mathcal{A}$ and $\mathcal{S}$, problems may arise only when $\mathcal{S}$ queries for the re-encryption of ciphertexts to meet the condition $(\bar{c}_i \mod 2^n) < 10^\ell$. However the number of queries is polynomial and the encryption function $c \mapsto E(K, c)$ is a random permutation, so the probability of such a collision is negligible.

Finally, throughout the simulation Step 5 of the algorithm has been ignored, always supposing that the check function outputs False (to be coherent the check function has to only check the simulated tokens already generated by $\mathcal{S}$).

In phases I and II, when $\text{check}(\texttt{token}_i) = \mathsf{True}$, $\mathcal{S}$ is able to follow the algorithm properly adjusting u_i and querying $\mathcal{C}$.

For the challenge phase instead we have to hope that $\text{check}(\texttt{token}) = \mathsf{False}$, which happens with non-negligible probability ρ_{q_1} (this probability depends on the number of queries q_1 made in Phase I). When $\text{check}(\texttt{token}) = \mathsf{True}$, we cannot simulate correctly, so $\mathcal{S}$ may directly guess randomly, and so the probability of guessing is $\frac{1}{2}$ in this case. Thus the advantage ϵ' of $\mathcal{S}$ is:

$$\epsilon' = \left| \left(\rho_{q_1} \left(\frac{1}{2} + \epsilon \right) + (1 - \rho_{q_1}) \frac{1}{2} \right) - \frac{1}{2} \right|$$

which is non-negligible since ϵ and ρ_{q_1} are non-negligible. Thus $\mathcal{S}$ has a non-negligible advantage winning the IND-CPA game for E. □

It is well-known that an encryption algorithm which is IND-CPA secure then it is secure against a *known-plaintext attack*, and even more so against a *ciphertext-only* attack. We now show similar results for our tokenization algorithm by proving some of our claimed requirements.

Corollary 1 *(***A1-A2-A3***) Let T be the Tokenization Algorithm described in Sect. 3, and let E be the block cipher used in Step 2 of T. If E is secure in an IND-CPA scenario then T satisfies A1, A2 and A3.*

Proof – A1

If T does not satisfy A1 then an attacker $\mathcal{A}$ can obtain a PAN from a token once she gets enough tokens corresponding to the same PAN. So we can suppose that $\mathcal{A}$ has access to an algorithm that takes in input a polynomial number N of `tokens` and with a non-negligible probability outputs

- X, if all the N `tokens` correspond to X;
- False, otherwise.

$\mathcal{A}$ tries the IND-CPA game for T and chooses $N-1$ pairs (X, u_i), with the same X, and sends them to $\mathcal{C}$, who responds with $\mathtt{token}_i = T(K, X, u_i)$. Then $\mathcal{A}$ passes to the Challenge Phase and chooses u_0^* and (X_1^*, u_1^*), with $X_1^* \neq X$ and sends (X_0^*, u_0^*), with $X_0^* = X$, and (X_1^*, u_1^*) to $\mathcal{C}$ that selects $\nu \in \{0,1\}$ at random and computes $\mathtt{token} = T(K, X_\nu^*, u_\nu^*)$. Then $\mathcal{C}$ sends `token` to $\mathcal{A}$. Then $\mathcal{A}$ runs her algorithm passing as inputs $\{\mathtt{token}_i\}_{1\le i\le N-1}$ and `token`, obtaining with non-negligible probability either $X_0^* = X$ or False and so she knows for sure and wins the IND-CPA game.

– A2

If A2 does not hold, then $\mathcal{A}$ has access to an algorithm that takes two inputs, a polynomial number N of token-to-PAN pairs and a token, and returns the PAN corresponding to the token with non-negligible probability. Thanks to our convention in Sect. 3, we can assume the algorithm inputs to be actually N pairs of type $(X_i, \mathtt{token}_i)$ (plus the single token $\mathtt{token}^*$) and the algorithm output to be the $X^* \in \mathbb{P}$ corresponding to $\mathtt{token}^*$. The adversary tries the IND-CPA game for T by choosing N random (X_i, u_i)'s and obtaining from C their corresponding tokens $\mathtt{token}_i$'s. Then she passes to the Challenge Phase and sends two random pairs: (X_0^*, u_0^*) and (X_1^*, u_1^*). The Challenger returns the token `token` corresponding to one of these. Then $\mathcal{A}$ runs her algorithm passing as inputs the pairs $(X_i, \mathtt{token}_i)$'s and `token`, obtaining with non-negligible probability the correct $X \in \{X_0^*, X_1^*\}$, winning thus the IND-CPA game.

– A3

If A3 does not hold, then we can assume that $\mathcal{A}$ has access to an algorithm that takes as input a polynomial number N of tokens and one additional input u^*, and that returns a pair $(X, \mathtt{token}^*)$, where $\mathtt{token}^*$ is the token corresponding to (X, u^*) with non-negligible probability. The adversary tries the IND-CPA game for T by choosing N random (X_i, u_i)'s and obtaining from C their corresponding tokens $\mathtt{token}_i$'s. Then she chooses a $u^* \in \mathbb{U}$ and passes it along with these tokens to her algorithm, obtaining X and $\mathtt{token}^*$. Then she passes to the Challenge Phase and sends pairs: (X, u^*) and a random pair. The Challenger returns the token `token` corresponding to one of these. The adversary easily wins the game just by checking if $\texttt{token} = \texttt{token}^*$. □

Theorem 2 *(***A4 for Tokenization Algorithm***) Let $K, K^* \in \mathbb{K}$, $X \in \mathbb{P}$ and $u \in \mathbb{U}$. If an attacker knows only u and the token $\texttt{token} = T(K, X, u)$, then she is able to compute $\texttt{token}^* = T(K^*, X, u)$ only with negligible probability.*

Proof The two tokens come directly from the encryption of the same string $M = f(u, X) \mid\mid [\overline{X}]_2^n$ with two different keys, except when the unlikely condition in Step 3 is met. So, with non-negligible probability, $\mathcal{A}$ is able to compute $E(K^*, M)$ from $E(K, M)$. This means that for a large portion of the plaintext space the two encryption functions are closely correlated, since from one it is possible to deduce the other without the need for decryption/re-encryption contradicting our first assumption on E, that is, that the set of its encryption function forms a random sample of the set of permutations acting on $(\mathbb{F}_2)^m$. □

Remark 1 The requirement GT5 in [13] "*The recovery of the original PAN should be computationally infeasible knowing only the token, a number of tokens, or a number of PAN/token pairs*" directly follows from A1 and A2.

5 Conclusion

Tokenization is a problem of practical interest, so we conclude giving an example of an instantiation with concrete cryptographic primitives and fixed length of the PAN, and we will analyze its efficiency and security.

Let us consider PANs with length $\ell = 16$, so $n = 54$, let us take AES-256 as the cipher E, and as the function f we take SHA-256 truncated to $128 - 54 = 74$ bits.

5.1 Security

Given the results given in [10], SHA-256 passed several statistical tests designed to verify "the absence of any detectable correlation between input and output and the absence of any detectable bias due to single bit changes in the input string". Therefore, it can be considered collision-resistant and so it satisfies our purposes (see, for instance, Step 5 of our algorithm).

Moreover, in [11], the authors showed that AES-256 passed statistical tests designed to verify the following properties:

- "the absence of any detectable correlation between plaintext/ciphertext pairs and the absence of any detectable bias due to single bit changes to a plaintext block"; and
- "the absence of any detectable deviations from randomness".

Therefore, we can consider AES-256 IND-CPA secure and so it satisfies the hypothesis of Theorem 1 and Corollary 1, and the randomness requirements of Theorem 2.

5.2 Efficiency

The probability that the condition at Step 3 is met is

$$p = \frac{2^{54} - 10^{16}}{2^{54}} \approx 0.445 \tag{5.1}$$

We have a geometric distribution, so the expected value of the number of iterations is:

$$\mathbb{E}_1 = \sum_{k=1}^{\infty} kp^{k-1}(1-p) \approx 1.801$$

thus on average less than 2 executions of AES-256 are needed to get to Step 5.

To quantify the probability to meet the condition of Step 5 we have to estimate the size of the database. A very generous upper bound is 10^9 PANs and 10^4 token per PAN (generating a new token for every transaction) for a total of 10^{13} entries in the database. Considering that there are $10^{16} - 1$ possible tokens, the probability to meet the condition is:

$$\rho = \frac{10^{13}}{10^{16} - 1} \approx 0.001$$

Again, the expected value of the number of iterations of the algorithm is:

$$\mathbb{E}_2 = \sum_{k=1}^{\infty} k\rho^{k-1}(1-\rho) \approx 1.001.$$

thus very rarely more than one execution of SHA-256 is needed.

Finally the expected value of the total number of executions of AES-256 is:

$$\mathbb{E}_1\mathbb{E}_2 \approx 1.803.$$

thus on average less than 2 executions of AES-256 are needed to get to Step 6.

Acknowledgements The authors are indebted to several people for their suggestions: Sandra Díaz, Patrick Harasser, Alessandro Tomasi and the anonymous referee.

References

1. Bellare, M., Ristenpart, T., Rogaway, P., Stegers, T.: Format-preserving encryption. In: Jacobson, M.J., Rijmen, V., Safavi-Naini, R. (eds.) Selected Areas in Cryptography–SAC 2009. Lecture Notes in Computer Science, vol. 5867, pp. 295–312. Springer, Heidelberg (2009)
2. Bellare, M., Rogaway, P., Spies, T.: The FFX mode of operation for format-preserving encryption (Draft 1.1). Manuscript (standards proposal) submitted to NIST (2010)
3. Black, J., Rogaway, P.: Ciphers with Arbitrary Finite Domains, pp. 114–130. Springer, New York (2002)

4. Brier, E., Peyrin, T., Stern, J.: BPS: a Format-Preserving Encryption proposal. Manuscript (standards proposal) submitted to NIST (2010)
5. Díaz-Santiago, S., Rodríguez-Henríquez, L.M., Chakraborty, D.: A cryptographic study of tokenization systems. Int. J. Inf. Secur. **15**(4), 413–432 (2016)
6. EMVCo: Payment Tokenisation Specification—Technical Framework, Version 1.0. Technical Report (2014)
7. Hoang, V.T., Morris, B., Rogaway, P.: An enciphering scheme based on a card shuffle. In: Safavi-Naini, R., Canetti, R. (eds.) Advances in Cryptology–CRYPTO 2012. Lecture Notes in Computer Science, vol. 7414, pp. 1–13. Springer, Heidelberg (2012)
8. Liskov, M., Rivest, R.L., Wagner, D.: Tweakable block ciphers. J. Cryptol. **24**(3), 588–613 (2011)
9. Morris, B., Rogaway, P., Stegers, T.: How to encipher messages on a small domain. In: Halevi, S. (ed.) Advances in Cryptology–CRYPTO 2009. Lecture Notes in Computer Science, vol. 5677, pp. 286–302. Springer, Heidelberg (2009)
10. NIST: Secure Hash Standard (SHS). FIPS Publication, Information Technology Laboratory National Institute of Standards and Technology, Gaithersburg, MD, pp. 180-4 (2015)
11. Rukhin, A., et al.: A Statistical Test Suite for the Validation of Random and Pseudo Random Number Generators for Cryptographic Applications. NIST Special Publication, Information Technology Laboratory National Institute of Standards and Technology, Gaithersburg, MD (2010)
12. SSC, P.: Information Supplement: PCI DSS Tokenization Guidelines, Version 2.0. Technical Report (2011)
13. SSC, P.: Tokenization Product Security Guidelines—Irreversible and Reversible Tokens, Version 1.0. Technical Report (2015)
14. SSC, P.: PCI DSS Requirements and Security Assessment Procedures, Version 3.2. Technical Report (2016)
15. Stefanov, E., Shi, E.: FastPRP: Fast Pseudo-Random Permutations for Small Domains. IACR Cryptology ePrint Archive, Report 2012/254 (2012), http://eprint.iacr.org/