A Cryptographic Study of Tokenization Systems

Sandra Díaz-Santiago, Lil María Rodríguez-Henríquez and Debrup Chakraborty

Department of Computer Science, CINVESTAV-IPN,
Av. IPN 2508 San Pedro Zacatenco, Mexico City 07360, Mexico
sdiaz@computacion.cs.cinvestav.mx,lrodriguez@computacion.cs.cinvestav.mx, debrup@cs.cinvestav.mx

Abstract. Payments through cards have become very popular in today's world. All businesses now have options to receive payments through this instrument, moreover most organizations store card information of its customers in some way to enable easy payments in future. Credit card data is a very sensitive information and theft of this data is a serious threat to any company. Any organization that stores credit card data needs to achieve payment card industry (PCI) compliance, which is an intricate process where the organization needs to demonstrate that the data it stores is safe. Recently there has been a paradigm shift in treatment of the problem of storage of payment card information. In this new paradigm instead of the real credit card data a token is stored, this process is called "tokenization". The token resembles the credit/debit card number but is in no way related to it. This solution relieves the merchant from the burden of PCI compliance in several ways. Though tokenization systems are heavily in use, to our knowledge, a formal cryptographic study of this problem has not yet been done. In this paper we initiate a study in this direction. We formally define the syntax of a tokenization system, and several notions of security for such systems. Finally, we provide some constructions of tokenizers and analyze their security in the light of our definitions.[1]

1 Introduction

In our digital age, credit cards have become a popular payment instrument. With increasing popularity of business through internet, every business requires to maintain credit card information of its clients in some form. Credit card data theft is considered to be one of the most serious threats to any business. Such a breach not only amounts to a serious financial loss to the business but also a critical damage to the "brand image" of the company in question.

The Payment Card Industry Security Standard Council (PCI SSC), which was founded by the major payment card brands, is an organization responsible for the development and deployment of various best practices in ensuring security of credit card data. In particular, PCI SSC has developed a standard called the PCI Data Security Standard (PCI DSS) [11] which specifies security mechanisms required to secure payment card data. PCI DSS dictates that organizations, which process card payments, must protect cardholder data when they store, transmit and process them. The actual requirements specified by PCI DSS are elaborate and complex. To obtain PCI compliance, a merchant needs to provide documentation on the usage and security policies regarding *all* sensitive information stored in its environment. PCI compliance is considered to be necessary for any business to acquire the confidence of its customers. Moreover, a business which has suffered theft of sensitive information while not being compliant can be subject to hefty amounts of fines from the government in some countries.

Traditionally credit card numbers have been used as a primary identifier in many business processes in the merchant sites. We quote from a document by Securosis [17]:

[1] A small part of this work appears in the Proceedings of International Conference on Security and Cryptography, SECRYPT 2014

As the standard reference key, credit card numbers are stored in billing, order management, shipping, customer care, business intelligence, and even fraud detection systems. Large retail organizations typically store credit card data in every critical business processing system.

Thus, in merchant sites, credit card numbers are scattered across their environment. This makes it very difficult for a merchant to formulate security policies and provide necessary documentation to obtain PCI compliance.

But, in most systems where credit card numbers are stored, the data itself is not required, and the system would function as well as before if the credit card numbers are replaced by some other information which would "look like" credit card numbers. This observation has lead to a paradigm shift in the way security of credit card numbers are viewed: instead of securing sensitive data wherever it is present it is easier to remove sensitive data from where it is not required. This basic paradigm has been implemented using *tokens*. Tokens are numbers which represent credit cards but are unrelated to the real credit card numbers.

There have been numerous industry white papers and similar documents which try to popularize tokenization and discuss about the possible solutions to the tokenization problem [16, 17, 19, 15]. PCI SSC has also formulated its guidelines regarding *tokenization* [12]. But to our knowledge a formal cryptographic analysis of the problem has not been done. Even it is not clear what basic cryptographic objects should be used and in what way, to achieve the goals of tokenization.

SMALL DOMAIN ENCRYPTION. One obvious solution for securing credit card numbers in a merchant site is to encrypt them. But as we stated, a typical merchant site heavily depends on the credit card numbers for its functioning, even it index data based on credit card numbers. Hence, a strict requirement for applying encryption is that the cipher should look like a credit card number, so that for using encryption one does not require to change the database fields where these numbers are stored. This necessity opened up an interesting problem. A typical credit card number consists of sixteen (or less) decimal digits, if this is treated as a binary string, is about 53 bits long. This is much less than the block size of a typical block cipher (say AES). Thus, direct application of a block cipher to encrypt would result in a considerable length expansion, and it would not be possible to encode the cipher into sixteen decimal digits.

The general problem was named by Voltage Security as *format preserving encryption* (FPE), which refers to an encryption algorithm which preserves the format of the message. Formally, if we consider $\mathcal{X}$ to be a message space which contains strings from an arbitrary alphabet satisfying certain format, $\mathcal{D}$ and $\mathcal{K}$ be finite sets called the tweak space and key space respectively, then a format preserving encryption scheme is formally defined to be a function $\mathsf{FP} : \mathcal{K} \times \mathcal{D} \times \mathcal{X} \to \mathcal{X}$, such that for every $d \in \mathcal{D}$ and $K \in \mathcal{K}$, $\mathsf{FP}_K(d, \cdot) : \mathcal{X} \to \mathcal{X}$ is a permutation. And FP should provide security as that of a tweakable strong pseudorandom permutation (SPRP). Designing such schemes for arbitrary $\mathcal{X}$ is a challenging and interesting problem. In particular given a SPRP on $\{0,1\}^n$, designing a SPRP for a message space $\{0,1\}^t$, where $t < n$ is difficult. There have been some interesting solutions to this problem, but none of them can be considered to be efficient [1, 2, 5, 7, 10, 18].

A credit card number encrypted by an FPE scheme can act as a token. Such a solution is also provided by Voltage Security [19]. To the best of our knowledge, this is the only solution to the tokenization problem with known cryptographic guarantees. But again, there does not exist a formal security model for tokenization, and it has been contested that a token which is an encryption of the credit card data may not be considered as a safe token as there exists a possibility that the token can be inverted to get the original data [17].

OUR CONTRIBUTION. We study the problem of tokenization from a cryptographic viewpoint, the main contributions of this work can be summarized as follows. We point out the basic needs for a tokenization

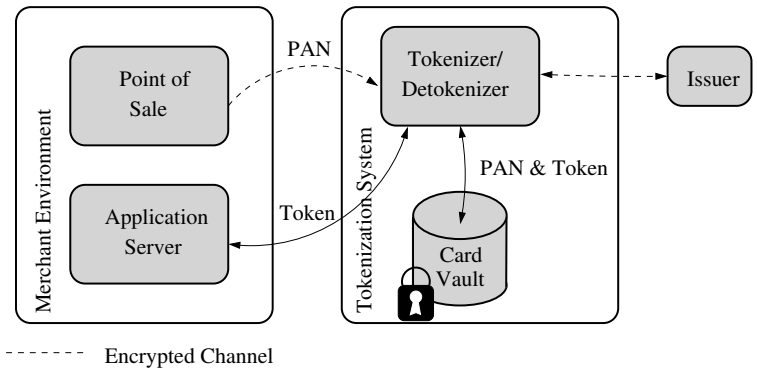

Fig. 1: Architecture of the tokenization system.

system, and develop a syntax for the problem. The syntax follows closely the recommendation of the PCI SSC, and it is general enough to accommodate various implementation options. Further, we develop a security model for the problem in line with concrete provable security. We propose three different security notions IND-TKR, IND-TKR-CV, and IND-TKR-KEY, which depend on three different threat models. We amply discuss the adequacy of these new notions of security in practical scenarios.

Finally, we propose some constructions of tokenization systems, and prove their security in the proposed security models. We propose three generic constructions namely TKR1, TKR2 and TKR2a and discuss how these constructions can be instantiated with existing cryptographic primitives. TKR1 is a construction which just uses a format preserving encryption to generate tokens. TKR2 and TKR2a are similar but both are very different from TKR1. In the constructions TKR2 and TKR2a we demonstrate how the problem of tokenization can be solved both securely and efficiently without using FPE. Both TKR2 and TKR2a uses off the shelf cryptographic primitives, in particular we show how to instantiate them using ordinary block ciphers, stream ciphers supporting initialization vectors (IV) and physical random number generators. We also prove security of our constructions in the proposed security models.

2 Tokenization Systems: Requirements and PCI DDS Guidelines

The basic architecture of a tokenization system is described in Fig. 1. In the diagram we show three separate environments: the merchant site, the tokenization system and the card issuer. The basic data objects of interest are the primary account number (PAN), which is basically the credit card number and the token which represents the PAN. A customer communicates with the merchant environment through the "point of sale", where the customer provides its PAN. The merchant sends the PAN to the tokenizer and gets back the corresponding token. The merchant may store the token in its environment. At the request of the merchant the tokenizer can *detokenize* a token and send the corresponding PAN to the card issuer for payments.

We show the tokenization system to be separated from the merchant environment, this is true in most situations today, as the merchants receive the service of tokenization from a third party. But it is also possible that the merchant itself implements its tokenization solution, and in that case, the tokenization system is a part of the merchant environment.

As described in [12], a *tokenization system* has the following components:

1. **A method for token generation:** A process to create a token corresponding to a *primary account number* (PAN). In [12] there is no specific recommendation regarding how this process should be implemented. Some of the mentioned options are encryption functions, cryptographic hash functions and random number generators.
2. **A token mapping procedure:** It refers to the method used to associate a token with a PAN. Such a method would allow the system to recover a PAN, given a token.
3. **Card-Vault:** It is a repository which usually will store pairs of PANs and tokens and maybe some other information required for the token mapping. Since it may contain PANs, it must be specially protected according the PCI DSS requirements.
4. **Cryptographic Key Management:** This module is a set of mechanisms to create, use, manage, store and protect keys used for the protection of PAN data and also data involved in token generation.

The PCI guidelines for tokenization are quite vague (this has been pointed out before in many places including [16]), and it is difficult to make out what properties tokens and tokenization systems should posses for functionality and security. We state two basic requirements for tokens and tokenization systems. We assume that tokenization is provided as a service, thus multiple merchants utilize the same system for their tokenization needs.

1. **Format Preserving:** The token should have the same format as that of the PANs, so that the stored PANs can be easily replaced by the tokens in the merchant environment. It has been said that in some scenarios it may be important that the tokens can be easily distinguished from that of the PANs. For example, most credit card numbers have a Luhn checksum [8] of zero. One can make tokens containing same number of digits as that of the PAN but the Luhn checksum should be 1. Such a distinguishing criteria may make audits easier.
2. **Uniqueness:** The token generation method should be deterministic. As stated before, the application software in the merchant side uses the PAN for indexing, thus the tokens for a specific PAN should be unique, i.e., if the same PAN is tokenized twice by the same merchant then the same token should be obtained. Moreover, in a specific merchant environment two different PANs should be represented by different tokens.

3 Cryptographic Preliminaries and Notations

GENERAL NOTATIONS. The set of all n bit strings would be denoted by $\{0,1\}^n$. We shall sometimes consider strings over an arbitrary alphabet AL, for $Y \in \mathsf{AL}^*$, by $|Y|_{\mathsf{AL}}$ we will denote the number of characters in the string Y. If $\mathsf{AL} = \{0,1\}$, and X is a string over AL, then we will use $|X|$ to denote the length of X in bits. If A is a finite set, then $\#A$ will denote the cardinality of A. If X, Y are strings, $X||Y$ will denote the concatenation of X and Y. For $A \in \{0,1\}^*$, $\mathsf{format}_n(X) = X_1||X_2||\ldots||X_m$, where $|X_i| = n$, for $1 \leq i \leq m-1$ and $0 \leq |X_m| < n$. If $X \in \{0,1\}^*$ is such that $|X| \geq \ell$, then $\mathsf{take}_\ell(X)$ will denote the ℓ most significant bits of X. For a non negative integer $i \leq 2^n - 1$, $\mathsf{bin}_n(i)$ will denote the n bit binary representation of i, and for any n-bit string X, $\mathsf{toInt}(X)$ will denote the integer represented by the string X.

For a finite set $\mathcal{S}$, $x \xleftarrow{\$} \mathcal{S}$ will denote x to be an element chosen uniformly at random from $\mathcal{S}$. We consider an adversary as a probabilistic algorithm that outputs a bit b. $\mathcal{A}^O \Rightarrow b$, will denote the fact that an adversary $\mathcal{A}$ has access to an oracle O and outputs b. In general an adversary would have other sorts of interactions, maybe with other adversaries and/or algorithms before it outputs, these would be clear from the context.

Unless mentioned otherwise, whenever we refer to resources of an adversary we would mean: the number of oracle queries made by it and its running time.

PSEUDORANDOM FUNCTIONS AND PERMUTATIONS. For finite sets A and B, by $\mathsf{Func}(A,B)$ we would mean the set of all functions mapping A to B, and $\mathsf{Perm}(A)$ would denote the set of all permutations on A (i.e., all bijective functions mapping A to A). If $A = \{0,1\}^n$ and $B = \{0,1\}^\ell$, then we would denote $\mathsf{Func}(A,B)$ by $\mathsf{Func}(n,\ell)$ and $\mathsf{Perm}(A)$ by $\mathsf{Perm}(n)$.

Consider the map $F : \mathbb{K} \times \mathbb{D} \to \mathbb{R}$ where $\mathbb{K}, \mathbb{D}, \mathbb{R}$ (commonly called keys, domain and range respectively) are all non-empty and $\mathbb{K}$ and $\mathbb{R}$ are finite. We view this map as representing a *family of functions* $F = \{F_K\}_{K \in \mathbb{K}}$, i.e., for each $K \in \mathbb{K}$, F_K is a function from $\mathbb{D}$ to $\mathbb{R}$ defined as $F_K(X) = F(K,X)$. For every $K \in \mathbb{K}$, we call F_K to be an instance of the family F.

Let $F : \mathbb{K} \times \mathbb{D} \to \mathbb{R}$ be a family of functions. We define the prf-advantage of an adversary $\mathcal{A}$ in breaking F as

$$\mathbf{Adv}^{\mathrm{prf}}_F(\mathcal{A}) = \left| \Pr[K \xleftarrow{\$} \mathbb{K} : \mathcal{A}^{F_k(\cdot)} \Rightarrow 1] - \Pr[\rho \xleftarrow{\$} \mathsf{Func}(\mathbb{D},\mathbb{R}) : \mathcal{A}^{\rho(\cdot)} \Rightarrow 1] \right|.$$

Similarly, if $E : \mathbb{K} \times \mathbb{D} \to \mathbb{D}$ is a family of permutations, we define the prp-advantage of an adversary $\mathcal{A}$ in breaking E as

$$\mathbf{Adv}^{\mathrm{prp}}_E(\mathcal{A}) = \left| \Pr[K \xleftarrow{\$} \mathbb{K} : \mathcal{A}^{E_K(\cdot)} \Rightarrow 1] - \Pr[\pi \xleftarrow{\$} \mathsf{Perm}(\mathbb{D}) : \mathcal{A}^{\pi(\cdot)} \Rightarrow 1] \right|.$$

A *tweakable enciphering scheme (TES)* is a function $\mathcal{E} : \mathbb{K} \times \mathbb{T} \times \mathcal{M} \to \mathcal{M}$ where $\mathbb{K}$ is the key space, $\mathbb{T}$ is the tweak set, and $\mathcal{M}$ is the message space and for every $K \in \mathbb{K}$ and $T \in \mathbb{T}$ we have that $\mathcal{E}(K,T,\cdot) = \mathcal{E}^T_K(\cdot)$ is a length preserving permutation. We define the $\widetilde{\mathrm{prp}}$ advantage of an adversary $\mathcal{A}$ as

$$\mathbf{Adv}^{\widetilde{\mathrm{prp}}}_{\mathcal{E}}(\mathcal{A}) = \left| \Pr[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{\mathcal{E}_K(\cdot,\cdot)} \Rightarrow 1] - \Pr[\pi \xleftarrow{\$} \mathsf{Perm}^{\mathbb{T}}(n) : \mathcal{A}^{\pi(\cdot,\cdot)} \Rightarrow 1] \right|,$$

where $\mathsf{Perm}^{\mathbb{T}}(\mathcal{M})$ is the set of length preserving tweak indexed permutations on $\mathcal{M}$. If the message space $\mathcal{M} = \{0,1\}^n$, then $\mathcal{E}$ is called a tweakable block cipher.

DETERMINISTIC CPA SECURE ENCRYPTION. Let $\mathbf{E} : \mathbb{K} \times \mathbb{T} \times \mathcal{M} \to \mathbb{C}$ be a deterministic encryption scheme with key space $\mathbb{K}$, tweak space $\mathbb{T}$, message space $\mathcal{M}$ and cipher space $\mathbb{C}$. We define the det-cpa advantage of any adversary $\mathcal{A}$, which does not repeat any query as

$$\mathbf{Adv}^{\text{det-cpa}}_{\mathbf{E}}(\mathcal{A}) = \left| \Pr[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{\mathbf{E}_K(\cdot,\cdot)} \Rightarrow 1] - \Pr[\mathcal{A}^{\$(\cdot,\cdot)} \Rightarrow 1] \right|,$$

where $\$(.,.)$ is an oracle, which on input $(d,x) \in \mathbb{T} \times \mathbb{M}$ returns a random string of the size of the cipher-text of x.

4 A Generic Syntax

A tokenization system has the following components:

1. $\mathcal{X}$, a finite set of *primary account numbers* or PAN's. $\mathcal{X}$ contains strings from a suitable alphabet with a specific format.
2. $\mathcal{T}$, a finite set of tokens. $\mathcal{T}$ also contains strings from a suitable alphabet with a specific format. It may be the case that $\mathcal{T} = \mathcal{X}$.
3. $\mathcal{D}$, a finite set of associated data. The associated data can be any data related to the business process [2].
4. CV, the card-vault. The card-vault is a repository where PAN's and tokens are stored, which may have a special structure for the ease of implementation of the token mapping procedure. In our syntax we shall use the CV to represent a state of the tokenization system. Whenever a new PAN is tokenized, possibly both the PAN and the generated token are stored in the CV, along with some additional data. Disregarding the structure of the CV, we consider that "basic" elements of CV comes from a set $\mathbb{Y}$.
5. $\mathcal{K}$, a key generation algorithm. A tokenization system may require multiple keys, all these keys are generated through the key generation algorithm.
6. TKR, the tokenizer. It is the procedure responsible for generating tokens from the PANs. We consider the tokenizer receives as input: the CV as a state, a key K generated by $\mathcal{K}$, some associated data d which comes from a set $\mathcal{D}$, and a PAN $x \in \mathcal{X}$. An invocation of TKR outputs a token t and also changes the CV. Thus, other than t, TKR also produces an element from $\mathbb{Y}$ which is used to update the CV. We use the square brackets to denote this interaction. We formally see TKR as a function $\mathsf{TKR}[\mathsf{CV}] : \mathcal{K} \times \mathcal{X} \times \mathcal{D} \to \mathcal{T} \times \mathbb{Y}$. For convenience, we shall implicitly assume the interaction of TKR with CV, and we will use $\mathsf{TKR}_K^{(1)}(x,d)$ and $\mathsf{TKR}_K^{(2)}(x,d)$ to denote the two outputs (in $\mathcal{T}$ and $\mathbb{Y}$, respectively) of TKR.
7. DTKR, the detokenizer which inverts a token to a PAN. As in case of tokenizer, we denote a detokenizer as a function $\mathsf{DTKR}[\mathsf{CV}] : \mathcal{K} \times \mathcal{T} \times \mathcal{D} \to \mathcal{X} \cup \{\perp\}$. For detokenization also, we shall implicitly assume its interaction with CV and for $K \in \mathcal{K}$, $d \in \mathcal{D}$ and $t \in \mathcal{T}$, we shall write $\mathsf{DTKR}_K(t,d)$ instead of $\mathsf{DTKR}[\mathsf{CV}](K,t,d)$.

   A tokenization procedure TKR_K should satisfy the following:

- For every $x \in \mathcal{X}$, $d \in \mathcal{D}$ and $K \in \mathcal{K}$, $\mathsf{DTKR}_K(\mathsf{TKR}_K^{(1)}(x,d),d) = x$.
- For every $d \in \mathcal{D}$, and $x, x' \in \mathcal{X}$, such that $x \neq x'$, $\mathsf{TKR}_K^{(1)}(x,d) \neq \mathsf{TKR}_K^{(1)}(x',d)$.

The second criteria focuses on a weak form of uniqueness. We want that two different PANs with the same associated data should produce different tokens, we do not disallow the case where two different PANs with different associated data have the same tokens. This requirement is clear if we consider the associated data d to be an identifier for a merchant. We do not want that a single merchant obtains the same token for two different PANs, but we do not care if two different merchants obtain the same token for two different PANs.

5 Security Notions

We define three different security notions, which consider three different attack scenarios:

1. IND-TKR: Tokens are only public. This represents the most realistic scenario where an adversary has access to the tokens only, and the card-vault data remains in-accessible.

[2] In our view, irrespective of other possible identifiers, the associated data should contain an identifier of the merchant. Thus if $d, d' \in \mathcal{D}$ are two associated data related to two different merchants, it should be that $d \neq d'$. For our notion of correctness this requirement for the associated data would be required.

Experiment Exp-IND-TKR$^{\mathcal{A}}$

1. The challenger selects $K \xleftarrow{\$} \mathcal{K}$
2. $Q \leftarrow \emptyset$.
3. **for** each query $(x,d) \in \mathcal{X} \times \mathcal{D}$ of $\mathcal{A}$,
4. the challenger computes $t \leftarrow \mathsf{TKR}_K^{(1)}(x,d)$, and returns t to $\mathcal{A}$.
5. $Q \leftarrow Q \cup \{(x,d)\}$
6. **until** $\mathcal{A}$ stops querying
7. $\mathcal{A}$ selects $(x_0,d_0),(x_1,d_1) \in (\mathcal{X} \times \mathcal{D}) \setminus Q$ and sends them to the challenger
8. The challenger selects a bit $b \xleftarrow{\$} \{0,1\}$ and returns $t \leftarrow \mathsf{TKR}_K^{(1)}(x_b,d_b)$ to $\mathcal{A}$.
9. The adversary $\mathcal{A}$ outputs a bit b'.
10. **If** $b = b'$ **output** 1 **else output** 0.

Experiment Exp-IND-TKR-CV$^{\mathcal{A}}$

1. The challenger selects $K \xleftarrow{\$} \mathcal{K}$
2. $Q \leftarrow \emptyset$.
3. **for** each query $(x,d) \in \mathcal{X} \times \mathcal{D}$ of $\mathcal{A}$,
4. the challenger computes $(t,c) \leftarrow \mathsf{TKR}_K(x,d)$, and returns (t,c) to $\mathcal{A}$.
5. $Q \leftarrow Q \cup \{(x,d)\}$
6. **until** $\mathcal{A}$ stops querying
7. $\mathcal{A}$ selects $(x_0,d_0),(x_1,d_1) \in (\mathcal{X} \times \mathcal{D}) \setminus Q$ and sends them to the challenger
8. The challenger selects a bit $b \xleftarrow{\$} \{0,1\}$ and returns $(t,c) \leftarrow \mathsf{TKR}_K(x_b,d_b)$ to $\mathcal{A}$.
9. The adversary $\mathcal{A}$ outputs a bit b'.
10. **If** $b = b'$ **output** 1 **else output** 0.

Experiment Exp-IND-TKR-KEY$^{\mathcal{A}}$

1. The challenger selects $K \xleftarrow{\$} \mathcal{K}$
2. $Q \leftarrow \emptyset$.
3. **for** each query $(x,d) \in \mathcal{X} \times \mathcal{D}$ of $\mathcal{A}$,
4. the challenger computes $t \leftarrow \mathsf{TKR}_K^{(1)}(x,d)$, and returns t to $\mathcal{A}$.
5. $Q \leftarrow Q \cup \{(x,d)\}$
6. **until** $\mathcal{A}$ stops querying
7. $\mathcal{A}$ selects $(x_0,d_0),(x_1,d_1) \in (\mathcal{X} \times \mathcal{D}) \setminus Q$ and sends them to the challenger
8. The challenger selects a bit $b \xleftarrow{\$} \{0,1\}$ and returns $t \leftarrow \mathsf{TKR}_K^{(1)}(x_b,d_b)$ and K to $\mathcal{A}$.
9. The adversary $\mathcal{A}$ outputs a bit b'.
10. **If** $b = b'$ **output** 1 **else output** 0.

Fig. 2: Experiments used in the security definitions: IND-TKR, IND-TKR-CV and IND-TKR-KEY

2. IND-TKR-CV : The tokens and the contents of the card-vault are public. This represents an extreme scenario where the adversary gets access to the card-vault data also.

3. IND-TKR-KEY : This represents another extreme scenario where the tokens and the keys are public.

We formally define the above three security notions based on the notion of indistinguishability, as is usually done for encryption schemes. Three experiments corresponding to the three attack scenarios discussed above are described in Figure 2. Each experiment represents an interaction between a challenger and an adversary $\mathcal{A}$. The challenger can be seen as the tokenization system, which in the beginning selects a random key from the key space and instantiates the tokenizer with the selected key. Then (in lines 3 to 6 of the experiments), the challenger responds to the queries of the adversary $\mathcal{A}$. The adversary $\mathcal{A}$ in each case queries with $(x,d) \in \mathcal{X} \times \mathcal{D}$, i.e., it asks for the outputs of the tokenizer for pairs of PAN and associated data of its choice. Finally, $\mathcal{A}$ submits two pairs of PANs and associated data to the challenger. The challenger selects one of the pairs uniformly at random and provides $\mathcal{A}$ with the tokenizer output for the selected pair. The task of $\mathcal{A}$ is to tell which pair was selected by the challenger. If $\mathcal{A}$ can correctly guess the selection of the challenger then the experiment outputs a 1 otherwise it outputs a 0. This setting is very similar to the way in which security of encryption schemes are defined for a chosen plaintext adversary.

The three experiments differ in what the adversary gets to see. In experiment Exp-IND-TKR$^{\mathcal{A}}$, $\mathcal{A}$, in response to its queries gets only the tokens and in Exp-IND-TKR-CV$^{\mathcal{A}}$ it gets both the tokens and the data that is stored in the card-vault. In Exp-IND-TKR-KEY$^{\mathcal{A}}$, $\mathcal{A}$ gets the tokens corresponding to its queries, and the challenger reveals the key to $\mathcal{A}$ after the query phase.

Definition 1. *Let* $\mathsf{TKR}[\mathsf{CV}] : \mathcal{K} \times \mathcal{X} \times \mathcal{D} \to \mathcal{T} \times \mathbb{Y}$ *be a tokenizer. Then the advantage of an adversary* $\mathcal{A}$ *in the sense of* IND-TKR*,* IND-TKR-CV *and* IND-TKR-KEY *are defined as*

$$\mathbf{Adv}^{\text{ind-tkr}}_{\mathsf{TKR}}(\mathcal{A}) = \left|\Pr[\text{Exp-IND-TKR}^{\mathcal{A}} \Rightarrow 1] - \frac{1}{2}\right|,$$

$$\mathbf{Adv}^{\text{ind-tkr-cv}}_{\mathsf{TKR}}(\mathcal{A}) = \left|\Pr[\text{Exp-IND-TKR-CV}^{\mathcal{A}} \Rightarrow 1] - \frac{1}{2}\right|,$$

$$\mathbf{Adv}^{\text{ind-tkr-key}}_{\mathsf{TKR}}(\mathcal{A}) = \left|\Pr[\text{Exp-IND-TKR-KEY}^{\mathcal{A}} \Rightarrow 1] - \frac{1}{2}\right|,$$

respectively.

From the definitions, it is obvious that IND-TKR-CV $\Longrightarrow$ IND-TKR and IND-TKR-KEY$\Longrightarrow$IND-TKR, but IND-TKR $\not\Longrightarrow$ IND-TKR-CV and IND-TKR$\not\Longrightarrow$IND-TKR-KEY. Thus IND-TKR-CV and IND-TKR-KEY are strictly stronger than IND-TKR.

ADEQUACY OF THE NOTIONS. We discuss some of the characteristics and limitations of the proposed definitions next.

1. IND-TKR refers to the basic security requirement for tokens. It adheres to the informal security notion for tokens as stated in the PCI DSS guideline for tokenization. It models the fact that tokens and PANs are un-linkable in a computational sense, if the key and card-vault are kept secret. Thus, if a merchant adopts a tokenization scheme provided by a third party, which is secure in the IND-TKR sense then this will probably relieve it from PCI compliance. As in this case the merchant does not own the card-vault or the keys, and the burden of security involved with the keys and the card-vault lies with the provider who offers the tokenization service.
2. The IND-TKR-CV is a stronger notion. If a tokenization system achieves this security, then it implies that tokens and PANs are un-linkable even with the knowledge of the card-vault. This in turn implies that the contents of the card-vault are not useful (in a computational sense) to derive a relation between PANs and tokens. Thus, it provides security both to the tokenization service provider and the merchant who use this service.
3. IND-TKR-KEY is a stronger form of the IND-TKR notion. Some public documents like [17] it has been stressed that encryption is not a good option for tokenization, as in theory there exists the possibility that a token can be inverted to obtain the PAN. If tokens are generated using a "secure" encryption scheme, then it is infeasible for any "reasonably efficient" adversary to invert the token without the knowledge of the key. But, this computational guarantee does not seem to be enough for users. The IND-TKR-KEY definition aims to model this paranoid situation, where linking the PANs with tokens becomes infeasible even with the knowledge of the key. Note in IND-TKR-KEY we still assume that the card-vault is inaccessible to an adversary.
4. All the definitions follow the style of a chosen plaintext attack. The definitions may be made stronger by giving the adversary additional power of obtaining PANs corresponding to tokens of its choice. But in this application, we think such stronger notions are not applicable.

In the following two sections we discuss two class of constructions for tokenizers. The first construction TKR1, is the trivial way to do tokenization using FPE. The other constructions (TKR2 and a variant TKR2a) presented in Section 7 are very different. For the later constructions our main aim is to by-pass the use of FPE schemes and use standard cryptographic schemes along with some encoding mechanism to achieve both security and the format requirements for arbitrarily formatted PANs/tokens.

6 Construction TKR1: Tokenization Using FPE

The construction TKR1 is described in Figure 3. TKR1 uses an FPE scheme $\mathsf{FP} : \mathcal{K} \times \mathcal{D} \times \mathcal{X} \to \mathcal{T}$ in an obvious way to generate tokens, assuming that $\mathcal{T} = \mathcal{X}$.

$\mathsf{TKR1}_k(x,d)$	$\mathsf{DTKR1}_k(t,d)$
1. $t \leftarrow \mathsf{FP}_k(d,x)$;	1. $x \leftarrow \mathsf{FP}_k^{-1}(d,t)$;
2. **return** (t, NULL)	2. **return** x

Fig. 3: The TKR1 tokenization scheme using a format preserving encryption scheme FP.

For security we assume that $\mathsf{FP}_k()$ is a tweakable pseudorandom permutation with a tweak space $\mathcal{D}$ and message space $\mathcal{T}$. Note, that this scheme does not utilize a card-vault and thus is stateless. The scheme is secure both in terms of IND-TKR and IND-TKR-CV. We formally state the security in the following theorem.

Theorem 1. *1. Let $\Psi = \mathsf{TKR1}$ be defined as in Figure 3, and $\mathcal{A}$ be an adversary attacking Ψ in the* IND-TKR *sense. Then there exists a $\widetilde{prp}$ adversary $\mathcal{B}$ such that*

$$\mathbf{Adv}^{\text{ind-tkr}}_{\Psi}(\mathcal{A}) \leq \mathbf{Adv}^{\widetilde{\text{prp}}}_{\mathsf{FP}}(\mathcal{B}),$$

where $\mathcal{B}$ uses almost the same resources as of $\mathcal{A}$.

2. Let $\Psi = \mathsf{TKR1}$ be defined as in Figure 3, and $\mathcal{A}$ be an adversary attacking Ψ in the IND-TKR-CV *sense. Then there exists a $\widetilde{prp}$ adversary $\mathcal{B}$ (which uses almost the same resources as of $\mathcal{A}$) such that*

$$\mathbf{Adv}^{\text{ind-tkr-cv}}_{\Psi}(\mathcal{A}) \leq \mathbf{Adv}^{\widetilde{\text{prp}}}_{\mathsf{FP}}(\mathcal{B}).$$

The first claim of the Theorem is an easy reduction where we design a prp adversary $\mathcal{B}$ which runs $\mathcal{A}$ and finally relate the advantages of the adversaries $\mathcal{A}$ and $\mathcal{B}$. The second claim directly follows from the first, as in the construction TKR1, there is no card-vault, thus an IND-TKR-CV adversary for TKR1 do not have any additional information compared to an IND-TKR adversary. The proof is provided in the Appendix A.1.

This scheme can be instantiated using any format preserving encryption scheme as described in [1, 5, 2, 10, 7, 18]. We discuss more on the impact of security and efficiency for specific instantiations in Section 8.

7 Construction TKR2: Tokenization Without Using FPE

Here we propose a class of constructions which avoids the use of format preserving encryption. Instead of a permutation on $\mathcal{T}$ which we use for the previous construction, we assume a primitive $\mathsf{RN}^{\mathcal{T}}()$, which when invoked (ideally) outputs a uniform random element in $\mathcal{T}$. This primitive can be keyed, which we will denote by $\mathsf{RN}^{\mathcal{T}}[k]()$, where k is a uniform random element of a pre-defined finite key space $\mathcal{K}$. $\mathsf{RN}^{\mathcal{T}}()$ can also be realized by using a keyed cryptographic primitive f_k, such instantiations would be more specifically denoted by $\mathsf{RN}^{\mathcal{T}}[f_k]()$. We define the rnd advantage of an adversary $\mathcal{A}$ attacking $\mathsf{RN}^{\mathcal{T}}()$ as

$$\mathbf{Adv}^{\text{rnd}}_{\mathsf{RN}}(\mathcal{A}) = \left| \Pr[k \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{\mathsf{RN}^{\mathcal{T}}[k]()} \Rightarrow 1] - \Pr[\mathcal{A}^{\$^{\mathcal{T}}()} \Rightarrow 1] \right|. \tag{1}$$

Where $\$^{\mathcal{T}}()$ is an oracle which returns uniform random strings from $\mathcal{T}$. The task of a rnd adversary $\mathcal{A}$ is to distinguish between $\mathsf{RN}^{\mathcal{T}}[k]()$ and its ideal counterpart when oracle access to these schemes are given to $\mathcal{A}$.

We describe a generic scheme for tokenization in Figure 4, which we call as TKR2 that uses $\mathsf{RN}^{\mathcal{T}}()$. For the description we consider that the card-vault CV is a collection of tuples, where each tuple has 3 components (x_1,x_2,x_3), where x_1,x_2,x_3 are the token, the PAN and associated data respectively. For a tuple $\mathsf{tup} = (x_1,x_2,x_3)$, we would use $\mathsf{tup}^{(i)}$ to denote x_i. Given a card-vault CV we also assume procedures to search for tuples in the CV. $\mathsf{SrchCV}(i,x)$ returns those tuples tup in CV such that $\mathsf{tup}^{(i)} = x$. If S be a set of tuples, then by $S^{(i)}$ we will denote the set of the i-th components of the tuples in S.

$\mathsf{TKR2}_k(x,d)$	$\mathsf{DTKR2}_k(t,d)$
1. $S_1 \leftarrow \mathsf{SrchCV}(2,x)$;	1. $S_1 \leftarrow \mathsf{SrchCV}(1,t)$;
2. $S_2 \leftarrow \mathsf{SrchCV}(3,d)$;	2. $S_2 \leftarrow \mathsf{SrchCV}(3,d)$;
3. $S \leftarrow S_1 \cap S_2$;	3. $S \leftarrow S_1 \cap S_2$;
4. **if** $S = \emptyset$	4. **if** $S = \emptyset$
5. $t \leftarrow \mathsf{RN}^{\mathcal{T}}[k]()$;	5. **return** $\perp$;
6. $c \leftarrow (t,x,d)$;	6. **else** let $\mathsf{tup} \in S$
7. $\mathsf{InsertCV}(c)$;	7. $x \leftarrow \mathsf{tup}^{(2)}$;
8. **else** let $\mathsf{tup} \in S$	8. **end if**
9. $t \leftarrow \mathsf{tup}^{(1)}$	9. **return** x
10. $c \leftarrow (t,x,d)$	
11. **end if**	
12. **return** (t,c)	

Fig. 4: The TKR2 tokenization scheme using a random number generator $\mathsf{RN}^{\mathcal{T}}()$.

As it is evident from the description in Figure 4, the detokenization operation is made possible through the data stored in the card-vault, and the detokenization is just a search procedure. Also, the determinism is assured by search.

Correctness: A limitation of the TKR2 scheme is that it may violate the property of uniqueness. It is not guaranteed that $\mathsf{TKR2}_k(x,d) \neq \mathsf{TKR2}_k(x',d')$ when $(x,d) \neq (x',d')$. As discussed before, for practical purposes a weak form of uniqueness is required, i.e., for $x \neq x'$, for any $d \in \mathcal{D}$, $\mathsf{TKR2}(x,d) \neq \mathsf{TKR2}(x',d)$. This requirement stems from the fact that a specific merchant with associated data d, may use the tokens as a primary key in its databases. Thus if $d \neq d'$, it can be tolerated that $\mathsf{TKR2}(x,d) = \mathsf{TKR2}(x',d')$, for any $x,x' \in \mathcal{X}$.

Let us assume that $\mathsf{RN}^{\mathcal{T}}()$ behaves ideally. If q unique tokens have been already generated with a specific associated data d, the probability that the $(q+1)^{th}$ token (generated with associated data d) is equal to any of the q previously generated tokens is given by $q/\#\mathcal{T}$. Thus, this probability of collision increases with the number of tokens already generated. If the total number of tokens generated by the tokenizer for a specific associated data is much smaller than the size of the token space (which will be the case in a practical scenario) this probability of collision would be insignificant [3]. But, still the uniqueness can be guaranteed

[3] According to [6] the total number of credit cards in 2012 from the four primary credit card networks (i.e. VISA, MasterCard, American Express, and Discover) was 546 millions ($\approx 2^{30}$). This can be considered as a reasonable upper bound for q. Assuming credit card numbers to be of 16 decimal digits, $\#\mathcal{T} = 10^{16} \approx 2^{53}$. These numbers leads to a collision probability of $1/2^{23}$ which is insignificant.

by an additional search as shown in Figure 5. Where $\mathsf{RN}^{\mathcal{T}}()$ is repeatedly invoked unless a token different from one already produced is obtained. Following the previous discussion, if q is small compared to $\#\mathcal{T}$, the expected number of repetitions required until a unique token is obtained would be small. The detokenization

$\mathsf{TKR2}_k(x,d)$
1. $S_1 \leftarrow \mathsf{SrchCV}(2,x)$;
2. $S_2 \leftarrow \mathsf{SrchCV}(3,d)$;
3. $S \leftarrow S_1 \cap S_2$;
4. **if** $S = \emptyset$
5. $\quad t \leftarrow \mathsf{RN}^{\mathcal{T}}[k]()$;
6. $\quad$ **if** $t \in S_2^{(1)}$ **go to** 4;
7. $\quad c \leftarrow (t,x,d)$;
8. $\quad \mathsf{InsertCV}(c)$;
9. **else** let $\mathsf{tup} \in S$
10. $\quad t \leftarrow \mathsf{tup}^{(1)}$
11. $\quad c \leftarrow (t,x,d)$
12. **end if**
13. **return** (t,c)

Fig. 5: Modified TKR2 to ensure uniqueness

corresponding to the modified tokenization scheme described in Figure 5 remains the same as described in Figure 4.

We formally specify the security of TKR2 later in this section, but it is easy to see that TKR2 is not secure in the IND-TKR-CV sense, as in the card-vault the PANs are stored in clear, hence if the card-vault is revealed then no security remains. This can be fixed by encrypting the tokens in the card-vault. To achieve security in terms of IND-TKR-CV, any CPA secure encryption can be used to encrypt the PANs stored in the card-vault. Note that for the encrypted PAN to be stored in the card-vault the format preserving requirement is not required. We modify TKR2 to TKR2a to achieve this. We discuss the details of TKR2 next.

Modifying TKR2 to TKR2a: For this modification, the structure of the card-vault is a bit different than for TKR2. In this case, each tuple contains two components. The first being the encryption of the token and the second the encryption of the PAN. We additionally use a deterministic CPA secure encryption (supporting associated data) scheme $\mathbf{E} : \mathbb{K} \times \mathcal{D} \times \mathcal{M} \rightarrow \mathbb{C}$, with key space $\mathcal{K}$, tweak (associated data) space $\mathcal{D}$ and message space $\mathcal{M}$. We assume that $\mathcal{T} = \mathcal{X} = \mathsf{AL}^*$, where AL is an arbitrary alphabet, such that $\#\mathsf{AL} \geq 2$. We fix $\mathsf{a}, \mathsf{b} \in \mathsf{AL}$ and define the message space $\mathcal{M}$ of $\mathbf{E}$ to be

$$\mathcal{M} = \{\mathsf{a}||x : x \in \mathcal{X}\} \bigcup \{\mathsf{b}||t : t \in \mathcal{T}\}.$$

Note that a and b are public quantities. The cipher space $\mathbb{C}$ can be arbitrary, i.e., it is not required that $\mathbb{C} = \mathcal{X}$, as the ciphers here would not be tokens but would be stored in the card-vault. We assume that $\mathcal{D}, \mathbb{C} \subseteq \{0,1\}^*$.

The tokenization scheme TKR2a described in Figure 6 uses the objects described above. The main difference with TKR2 is that pairs of token and PAN are stored in the card-vault in the encrypted form. An important characteristic of the way the encryption is applied is that the inputs are differently encoded in case of a token and a PAN. This ensures the even if a PAN and a token are the same, they produce different ciphertexts.

$\mathsf{TKR2a}_{k_1,k_2}(x,d)$
1. $z \leftarrow \mathbf{E}_{k_1}(d, \mathsf{a}||x)$;
2. $S \leftarrow \mathsf{SrchCV}(2,z)$;
3. **if** $S = \emptyset$,
4. **do**
5. $t \leftarrow \mathsf{RN}^{\mathcal{T}}[k_2]()$;
6. $t' \leftarrow \mathbf{E}_{k_1}(d, \mathsf{b}||t)$;
7. **while** $\mathsf{SrchCV}(1,t') \neq \emptyset$;
8. $c \leftarrow (t',z)$;
9. $\mathsf{InsertCV}(c)$;
10. **else** let $\mathsf{tup} \in S$
11. $t'' \leftarrow \mathsf{tup}^{(1)}$
12. $t' \leftarrow \mathbf{E}_{k_1}^{-1}(d,t')$
13. $c \leftarrow \mathsf{tup}$;
14. parse t' as $\mathsf{b}||t$;
15. **end if**
16. **return** (t,c)

$\mathsf{DTKR2a}_{k_2}(t,d)$
1. $t' \leftarrow \mathbf{E}_{k_1}(d, \mathsf{b}||t)$;
2. $S \leftarrow \mathsf{SrchCV}(1,t')$;
3. **if** $S = \emptyset$
4. **return** $\perp$;
5. **else** let $\mathsf{tup} \in S$
6. $z \leftarrow \mathsf{tup}^{(2)}$;
7. $x' \leftarrow \mathbf{E}_{k_1}^{-1}(d,z)$;
8. parse x' as $\mathsf{a}||x$;
9. **end if**
10. **return** x

Fig. 6: The TKR2a tokenization scheme.

7.1 Realizing $\mathsf{RN}^{\mathcal{T}}[k]$

The heart of the procedures TKR2 and TKR2a is the keyed primitive $\mathsf{RN}^{\mathcal{T}}[k]$, which can be realized by standard cryptographic objects. We discuss here a specific realization which uses a pseudorandom function $f : \mathcal{K} \times \mathbb{Z}_N \rightarrow \{0,1\}^L$, where L and N are sufficiently "large", the exact requirements for N and L will become clear later. We call the construction $\mathsf{RN}[f_k]()$ and it is shown in Figure 7.

For the construction shown in Figure 7, we assume that $\mathcal{T}$ contains strings of fixed length μ from an arbitrary alphabet AL. Let $\#\mathsf{AL} = \ell$, and $\lambda = \lceil \lg \ell \rceil$. Let $\sigma : \mathsf{AL} \rightarrow \{0,1,2,\ldots,\ell-1\}$ be a fixed bijection. The variable *cnt* can be considered as a state of the algorithm and it maintains its values across invocations. The basic idea behind the algorithm is to generate a "long" binary string using $f_k(cnt)$ and divide the string

$\mathsf{RN}[f_k]()$
1. $X \leftarrow f_k(cnt)$;
2. $X_1||X_2||\ldots||X_m \leftarrow \mathsf{format}_{\lambda}(X)$;
3. $Y \leftarrow \varepsilon$; (empty string)
4. $i \leftarrow 1$;
5. **while** $|Y|_{\mathsf{AL}} \neq \mu$,
6. **if** $\mathsf{toInt}(X_i) < \ell$,
7. $Y \leftarrow Y||\sigma^{-1}(\mathsf{toInt}(X_i))$;
8. $i \leftarrow i+1$;
9. **end while**
10. $cnt \leftarrow cnt + 1$;
11. **return** Y;

Fig. 7: Construction of $\mathsf{RN}()$ using a pseudorandom function $f_k()$.

into blocks of λ bits. If the integer corresponding to a block is less than ℓ then it is accepted otherwise it is discarded. The accepted blocks are encoded as elements in AL.

Choosing L and N: Let us define, $p = \Pr[y \xleftarrow{\$} \{0,1\}^\lambda : \mathsf{toInt}(y) < \ell] = \frac{\ell}{2^\lambda} > \frac{1}{2}$. Thus, if we assume that the output of $f_k()$ is uniformly distributed then an X_i passes the test in line 6 (of Figure 7) with probability p. Thus the expected number of times the while loop will run is at most 2μ. Thus, $L = 3\mu\lambda$, will be sufficient for all practical purposes.

Note that each invocation of $\mathsf{RN}[f_k]()$ increases the value of *cnt* by 1. Thus the value of N should be a conservative upper bound on the number of times $\mathsf{RN}[f_k]()$ needs to be invoked. $N = 2^{80} - 1$, should be sufficient for all practical purposes.

If f_k is a PRF then $\mathsf{RN}^{\mathcal{T}}[f_k]$ is secure in the rnd sense. We formally state this security property in the following theorem.

Theorem 2. *Let $\mathcal{A}$ be an arbitrary adversary attacking $\mathsf{RN}[f_k]$ (as described in Figure 7) in the rnd sense. Then there exists a prf adversary $\mathcal{B}$ (which uses almost the same resources as of $\mathcal{A}$) such that*

$$\mathbf{Adv}^{\mathrm{rnd}}_{\mathsf{RN}[f_k]()}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{prf}}_{f_k}(\mathcal{B}). \tag{2}$$

This theorem asserts that as long as $f_k()$ is a PRF, the construction achieves the desired security in the rnd sense.

7.2 Candidates for $f_k()$:

$f_k()$ can be instantiated through standard symmetric key primitives. We discuss three options below:

1. *Stream cipher*: Modern stream ciphers, such as those in the eStream [14] portfolio, take as input a short secret key K and a short initialization vector (IV) and produce a "long" and random looking string of bits. Let $\mathsf{SC}_K : \{0,1\}^\ell \to \{0,1\}^L$ be a stream cipher with IV, i.e., for every choice of K from a certain predefined key space $\mathcal{K}$, SC_K maps an ℓ-bit IV to an output string of length L bits. The basic idea of security is that for a uniform random K and for distinct inputs $IV_1, \ldots, IV_q$, the strings $\mathsf{SC}_K(IV_1), \ldots, \mathsf{SC}_K(IV_q)$ should appear to be independent and uniform random to an adversary. This is formalized by requiring a stream cipher to be a PRF. See [3] for further discussion on this issue. Thus, a stream cipher with the above security guarantees can be used to instantiate f_k.
2. *Block cipher*: A block cipher $E : \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$ can also be used to construct f_k as follows.

$f_k(cnt)$
1. $m \leftarrow \lceil L/n \rceil$;
2. $Y \leftarrow \mathsf{bin}_n(cnt)$;
3. $W \leftarrow E_k(Y)$;
4. $Z \leftarrow E_k(W) || E_k(W \oplus \mathsf{bin}_n(1)) || \cdots$
   $\cdots || E_k(W \oplus \mathsf{bin}_n(m-1))$;
5. **return** $\mathsf{take}_L(Z)$

The above construction is same as the counter mode of operation, and if E_k is assumed to be a PRF then f_k as constructed above is also a PRF, in particular it is easy to verify the following holds

Proposition 1. *Let $\mathcal{B}$ be an arbitrary prf adversary attacking $f_k()$ who asks at most q queries, then one can construct a prf adversary $\mathcal{B}'$ for $E_K()$ such that, $\mathcal{B}'$ asks at most mq queries and*

$$\mathbf{Adv}_f^{\text{prf}}(\mathcal{B}) \leq \mathbf{Adv}_E^{\text{prf}}(\mathcal{B}') + \frac{m^2q^2}{2^n}.$$

3. *True random number generator:* We end this discussion with another possible interesting instantiation of $\mathsf{RN}()$. The specific construction that we depicted in Figure 7 basically uses a stream of random bits generated through a pseudorandom function. Currently there has been a lot of interest in designing physical true random number generators. Such generators harvests entropy from its "environment" and generates random streams with some post processing. It has been claimed that such generators are "true random number generators" (TRNG). Such a generator can be used to design $\mathsf{RN}()$ as in Figure 7 by replacing $f_k()$ with a TRNG, and by selecting suitable blocks from the generated stream according to the format requirements of $\mathcal{T}$. As a TRNG is key-less, thus this would lead to a key-less construction of RN, we call such an instantiation as $\mathsf{RN}[\mathsf{TR}]$. As such a generator gives us "true randomness", hence for any adversary $\mathcal{A}$, $\mathbf{Adv}^{\text{rnd}}_{\mathsf{RN}[\mathsf{TR}]} = 0$.

From now onwards, where it is necessary, we will denote $\mathsf{TKR2}$ instantiated with $\mathsf{RN}[f_k]$ and $\mathsf{RN}[\mathsf{TR}]$ by $\mathsf{TKR2}[f_k]$ and $\mathsf{TKR2}[\mathsf{TR}]$ respectively. Similar convention would be followed for $\mathsf{TKR2a}$.

7.3 Realizing $\mathbf{E}_k(d,x)$

As discussed previously, $\mathbf{E_k}(\cdot,\cdot)$ is used to encrypt the PAN, and the encryption is stored in the card-vault within the tokenization system. We do not require this encryption to be format preserving. Here we discuss two instantiations of $\mathbf{E}$ using a secure block cipher E. If the block length of E is n, then both the proposed constructions have $\{0,1\}^n$ as their cipher space, and $\mathcal{X}$ and $\mathcal{D}$ as their message space and tweak space, respectively. For the constructions we assume some restrictions on $\mathcal{X}$ and $\mathcal{D}$, but these restrictions would be satisfied in most practical scenarios.

Let $E : \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$ be a block cipher. As we defined before, let $\mathcal{X}$ contain strings of fixed length μ from an arbitrary alphabet AL where $\#\mathsf{AL} = \ell$ and $\lambda = \lceil \lg \ell \rceil$. Let $\#\mathcal{D} = \ell_1$ and $\lambda_1 = \lceil \lg \ell_1 \rceil$. Let n_1 and n_2 be positive integers such that $n_1 \geq \mu\lambda$, $n_2 \geq \lambda_1$ and $n_1 + n_2 = n$. Note that for practical choice of AL, $\mathcal{D}$, μ and n, such n_1, n_2 can be selected. Let $\mathsf{pad}_{\mathcal{X}} : \mathcal{X} \to \{0,1\}^n$, $\mathsf{pad}_{\mathcal{D}} : \mathcal{D} \to \{0,1\}^n$, $\mathsf{pad}_1 : \mathsf{AL}^\mu \to \{0,1\}^{n_1}$ and $\mathsf{pad}_2 : \mathcal{D} \to \{0,1\}^{n_2}$ be injective functions.

$\mathbf{E1}_K(d,x)$	$\mathbf{E1}_K^{-1}(d,z)$	$\mathbf{E2}_K(d,x)$	$\mathbf{E2}_K^{-1}(d,z)$
1. $z_1 \leftarrow \mathsf{pad}_1(x)$;	1. $y \leftarrow E_K^{-1}(z)$;	1. $z_1 \leftarrow \mathsf{pad}_{\mathcal{X}}(x)$;	1. $y \leftarrow E_K^{-1}(z)$;
2. $z_2 \leftarrow \mathsf{pad}_2(d)$;	2. $z_1 \leftarrow \mathsf{take}_{n_1}(y)$;	2. $z_2 \leftarrow \mathsf{pad}_{\mathcal{D}}(d)$;	2. $z_2 \leftarrow \mathsf{pad}_{\mathcal{D}}(d)$;
3. $z \leftarrow E_K(z_1 \| z_2)$;	3. $x \leftarrow \mathsf{pad}_1^{-1}(z_1)$;	3. $z \leftarrow E_K(z_1 \oplus E_K(z_2))$;	3. $x \leftarrow y \oplus E_K(z_2)$;
4. **return** z	4. **return** x	4. **return** z	4. **return** x

Fig. 8: The two instantiations of $\mathbf{E}_K$.

The two different proposed instantiations of $\mathbf{E}$ are shown in Figure 8. Both the constructions uses a block cipher with a block length of n, and the padding functions defined above. In $\mathbf{E1}$, the message x and

the associated data d are suitably formatted to a n bit string and this formatted string is encrypted using the block cipher. **E2** is same as the construction of a tweakable block cipher proposed in [9]. If E_K is a secure block cipher in the prf sense then both $\mathbf{E1}_K$ and $\mathbf{E2}_K$ are det-cpa secure, we state this formally next.

Proposition 2. *Let $\mathcal{A}$ be an arbitrary det-cpa adversary attacking* **E1***, who asks at most q queries, never repeats a query, and runs for time at most T, then there exists a prf adversary $\mathcal{B}$ such that*

$$\mathbf{Adv}^{\text{det-cpa}}_{\mathbf{E1}}(\mathcal{A}) \leq \mathbf{Adv}^{\text{prf}}_{E}(\mathcal{B}),$$

and $\mathcal{B}$ also asks exactly q queries and runs for time $O(T)$.

Proposition 3. *Let $\mathcal{A}$ be an arbitrary det-cpa adversary attacking* **E2***, who asks at most q queries, never repeats a query, and runs for time at most T, then there exists a prf adversary $\mathcal{B}$ such that*

$$\mathbf{Adv}^{\text{det-cpa}}_{\mathbf{E2}}(\mathcal{A}) \leq \mathbf{Adv}^{\text{prf}}_{E}(\mathcal{B}) + \frac{2q^2}{2^n},$$

and $\mathcal{B}$ also asks exactly q queries and runs for time $O(T)$.

The above propositions suggests that **E1** has a better security bound compared to **E2**, and for **E2** two block cipher calls are required for each encryption, whereas only a single block cipher call is required for **E1**. The formatting requirements are more stringent for **E1**, where as **E2** can be applied to any message space $\mathcal{X}$ and tweak space $\mathcal{D}$, where $\#\mathcal{X} \leq 2^n$ and $\#\mathcal{D} \leq 2^n$.

7.4 Security of TKR2 and TKR2a

The following three theorems specify the security of TKR2 and TKR2a.

Theorem 3. *Let $\Psi \in \{\mathsf{TKR2}, \mathsf{TKR2a}\}$ and $\mathcal{A}$ be an adversary attacking Ψ in the* IND-TKR *sense. Then there exists a RND adversary $\mathcal{B}$ (which uses almost the same resources as of $\mathcal{A}$) such that*

$$\mathbf{Adv}^{\text{ind-tkr}}_{\Psi}(\mathcal{A}) \leq \mathbf{Adv}^{\text{rnd}}_{\mathsf{RN}}(\mathcal{B})$$

Theorem 4. *Let $\Psi = \mathsf{TKR2a}$ and $\mathcal{A}$ be an adversary attacking Ψ in the* IND-TKR *sense. Then there exist adversaries $\mathcal{B}$ and $\mathcal{B}'$ (which use almost the same resources as of $\mathcal{A}$) such that*

$$\mathbf{Adv}^{\text{ind-tkr-cv}}_{\Psi}(\mathcal{A}) \leq \mathbf{Adv}^{\text{rnd}}_{\mathsf{RN}}(\mathcal{B}) + \mathbf{Adv}^{\text{det-cpa}}_{\mathbf{E}}(\mathcal{B}') + \frac{(q+1)^2}{2^{s+1}}$$

where s is the size of the shortest element in the cipher space of **E**.

Theorem 5. *Let $\Psi \in \{\mathsf{TKR2[TR]}, \mathsf{TKR2a[TR]}\}$ and $\mathcal{A}$ be an arbitrary adversary attacking Ψ in the* IND-TKR-KEY *sense. Then*

$$\mathbf{Adv}^{\text{ind-tkr-key}}_{\Psi}(\mathcal{A}) = 0$$

The proofs use standard reductionist arguments, we discuss them in Appendix A.

	IND-TKR	IND-TKR-CV	IND-TKR-KEY
TKR1	$\surd$	$\surd$	
TKR2$[f_k]$	$\surd$		
TKR2[TR]	$\surd$		$\surd$
TKR2a$[f_k, \mathbf{E}]$	$\surd$	$\surd$	
TKR2a[TR, $\mathbf{E}$]	$\surd$	$\surd$	$\surd$

Table 1: Summary of Security

8 Discussions

SECURITY. The security properties of the various schemes as stated in the previous security theorems are summarized in Table 1. The security theorems in all cases are to be interpreted carefully. We note down some relevant issues below.

In TKR1 the security is gained from the security of the format preserving encryption. The scheme FP used in TKR1 is required to be a tweakable pseudorandom permutation with the message/cipher space $\mathcal{T}$ and the tweak space $\mathcal{D}$. It is important to note that various instantiations of FP can give different security guarantees. Most of the known FPE schemes can only ensure security (in provable terms) when the number of queries made by an adversary is highly restricted. For example, the security claim of the scheme based on Feistel networks discussed in [4] becomes vacuous when the number of queries exceeds $2^{\#\mathcal{T}/4}$, whereas the scheme in [10] can tolerate up to $2^{\#\mathcal{T}-\varepsilon}$ queries where ε is inversely related to the number of rounds in the construction. Some recent constructions in [7, 13] achieve much better bounds, specially in [13] almost $\#\mathcal{T}$ queries can be tolerated for the bound to be meaningful. As $\#\mathcal{T}$ can be much smaller than the typical domain of a block cipher (2^n, for $n = 128$), thus the exact security guarantees are important in this context. Note, that for a typical scenario we consider credit card numbers of sixteen decimal digits then $\#\mathcal{T} \approx 2^{53}$.

In the construction of TKR2 and TKR2a the security bounds are better. If RN$[f_k]$ is instantiated as in Figure 7, and in turn f_k is constructed using a block cipher, then using Proposition 1 and Theorem 3, for any IND-TKR adversary $\mathcal{A}$ who asks at most q queries, we have

$$\mathbf{Adv}^{\text{ind-tkr}}_{\Psi}(\mathcal{A}) \leq \frac{m^2 q^2}{2^n} + \varepsilon_q,$$

where $\Psi \in \{\mathsf{TKR2}, \mathsf{TKR2a}\}$ and ε_q is the maximum prf advantage of any adversary (who asks at most q queries) in attacking the block cipher E. Note that, n is the block length of the block cipher used to construct f_k. And m depends on $\#\mathcal{T}$, as per the description of the block cipher based construction in Section 7.2, $m = L/n$, and we discussed that it would be enough if we take $L = 3\mu\lambda$, where μ is the length of each token where the tokens are treated as strings in AL and $\lambda = \lceil \lg \#\mathsf{AL} \rceil$. Thus, the security bound is less sensitive on $\#\mathcal{T}$. The bound only becomes vacuous when mq is of the order of $2^{n/2}$. A similar bound holds for $\mathbf{Adv}^{\text{ind-tkr-cv}}_{\mathsf{TKR2a}[f_k]}(\mathcal{A})$, when a block cipher based construction for f_k is used.

The IND-TKR-KEY definition is meant to model the property of independence of the tokens with the keys, and this represents a quite strong notion of security. The constructions TKR2$[f_k]$ and TKR2a$[f_k]$ do not achieve this security. But TKR2[TR] and TKR2a[TR] achieve security in the IND-TKR-KEY sense as here we are assuming an instantiation by a "true" random number generator.

EFFICIENCY. The efficiency of TKR1 depends on the efficiency of the FP scheme. As discussed there are various ways to instantiate FP with varying amount of security and efficiency. Also, most schemes with provable guarantees are far inefficient than standard block ciphers.

The efficiency of TKR2 and TKR2a would be dominated by the search procedure. Asymptotically, if $\#\mathcal{T} = N$, then tokenization and detokenization would take $O(\lg N)$ time. But the hidden constant would depend on how efficiently the search has been implemented and how powerful the machine is (mainly in terms of memory). We discussed more about this in Section 9.

9 Experimental Results

We performed some preliminary experiments to determine the efficiency and functionalities of the proposed constructions in a practical environment. All experiments reported used the following computing resources:

CPU: Four-core i5-2400 Intel processor (3.1GHz).
OS: Ubuntu 12.04.4 LTS.
DataBase: PostgreSQL 9.2.6
Compiler: gcc 4.7.3

We implemented both $\mathsf{TKR2}[f_k]$ and $\mathsf{TKR2a}[f_k]$, instantiated with $\mathsf{RN}[f_k]$ (described in Figure 7), where f_k was instantiated with block cipher based construction described in Section 7.2.

We implemented the card-vault in a PostgresSQL database. For TKR2 we considered the card-vault to be a relation with three attributes: the token (TKN), the associated data (ASD) and the PAN. For this construction the primary key is composed by the token and the associated data. For TKR2a we considered the card-vault to be a relation with two attributes EPAN and ETKN, representing the encrypted PAN and token respectively. We encrypt these data using the construction **E1** described in Section 7.3. In this case ETKN was considered as the primary key.

For implementation of $f_k()$ we used AES with 128 bit key, and the implementation was done by using the new Intel AES-NI instruction set, which provides a very efficient and secure implementation of the AES. We assumed that $\mathcal{X}$ contains strings of 16 characters where each character is a decimal digit, and $\mathcal{T} = \mathcal{X}$. Thus, in accordance to our notations introduced before, we had $\mu = 16$, $\mathsf{AL} = \{0,1\ldots,9\}$, thus $\lambda = \lceil \lg(\#\mathsf{AL}) \rceil = 4$, and $\mathcal{X} = \mathcal{T} = \mathsf{AL}^{\mu}$.

The reported times are based on an $-$O3 optimized code. The time was measured by first measuring the number of cycles necessary for a specific operation using the `rdtsc` instruction. This cycle counts we converted to real time using the processor frequency.

We summarize our experiments and the results below:

1. The first experiment was to verify how many block cipher calls are necessary for each call of $f_k()$ and the efficiency of $\mathsf{RN}^{\mathcal{T}}[f_k]$. In Section 7, we discussed that if the range of f_k is $\{0,1\}^L$, then $L \leq 3\lambda\mu$ would be sufficient. Note, that the number of block cipher calls required for each invocation of f_k is $m = \lceil L/\lambda \rceil$. We made 1000000 independent calls to f_k, and in all cases, in each call we required at most two block cipher calls. In fact in only 5% of the cases two calls were necessary. In all others only one call was sufficient. The average time required for each invocation of $\mathsf{RN}^{\mathcal{T}}[f_k]$ was 0.1 microseconds.
2. The second experiment was to see if TKR2 implemented without the uniqueness test (as described in Figure 4) would be sufficient. Again, we generated 1000000 tokens using TKR2 and they were all

unique. Thus, in a practical scenario, where the card-vault would be stored in a database, the uniqueness test (as included in the description in Figure 5) is not required to be explicitly included. Once a token is generated and when the system tries to insert it in the database, if the uniqueness condition is violated then the database would generate an error message, and then the process may be repeated until a unique token is generated.

3. Finally we measured efficiency of the tokenization procedures TKR2 and TKR2a. In Table 2 we summarize the results, which are described as below:
   - **Run1** denotes the average time required to generate one token, including the insertion in the card-vault. But here primary keys in the card-vault relations are not specified, i.e., this run does not do any uniqueness test. The average is computed over 1000000 tokens.
   - **Run2** denotes the scenario where the primary keys are specified, i.e., the database checks for the uniqueness. As it is obvious, in this case the time required to tokenize (including the insertions in the card-vault) would increase with the current size of the card-vault. To measure this difference we divided this run into five different runs which we call **Run2a**, **Run2b**,$\cdots$, **Run2e**. For **Run2a** we started with an empty card-vault, and generated 1000000 tokens. In **Run2b** we started with a card-vault already containing 1000000 tokens, and we generated 1000000 more tokens. Similarly, in runs **Run2c**, **Run2d** and **Run2e**, we started with a card-vault containing 2000000, 3000000 and 4000000 tokens, respectively. In each run we generated 1000000 more tokens. The Table 2 shows the average time required for generating one token for each scenario.

Experiment	**Time**(ms)	
	TRK2	TKR2a
Run1	0.19	0.26
Run2a	0.30	0.37
Run2b	0.83	0.96
Run2c	1.27	1.30
Run2d	1.49	1.52
Run2e	1.69	1.98

Table 2: Summary of the experimental results: The descriptions of **Run1**, **Run2a**,$\cdots$ **Run2e** are provided in the text.

The basic component for both TKR2 and TKR2a is the procedure $\mathsf{RN}^{\mathcal{T}}$, as mentioned, a call to $\mathsf{RN}^{\mathcal{T}}$, costs only 0.1 micro seconds. But the times reported in Table 2 (which are in milliseconds) are more realistic, and it shows that the database insertions dominate the cost of tokenization. Thus, further optimization in this regard may be possible. But, still our experimental results confirms that the schemes proposed in this work can be implemented and used in a real tokenization environment.

10 Conclusion

We studied the problem of tokenization from a cryptographic viewpoint. We proposed a syntax for the problem and also formulated three different security definitions. These new definitions may help in analyzing existing tokenization systems. We also proposed three constructions for tokenization: TKR1, TKR2 and TKR2a. The constructions TKR2 and TKR2a are particularly interesting, as they demonstrate that tokenization can be achieved without the use of format preserving encryption. We analyzed all the constructions in

light of our security definitions and also provided some preliminary experimental results. We plan to perform a more rigourous efficiency comparison of the proposed schemes with the existing FPE schemes in near future.

References

1. 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*, volume 5867 of *Lecture Notes in Computer Science*, pages 295–312. Springer, 2009.
2. M. Bellare, P. Rogaway, and T. Spies. The FFX mode of operation for format-preserving encryption. NIST submission, February 2010. http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/ffx/ffx-spec.pdf.
3. C. Berbain and H. Gilbert. On the security of IV dependent stream ciphers. In A. Biryukov, editor, *FSE*, volume 4593 of *Lecture Notes in Computer Science*, pages 254–273. Springer, 2007.
4. J. Black and P. Rogaway. Ciphers with arbitrary finite domains. In B. Preneel, editor, *CT-RSA*, volume 2271 of *Lecture Notes in Computer Science*, pages 114–130. Springer, 2002.
5. E. Brier, T. Peyrin, and J. Stern. BPS: a format-preserving encryption proposal. NIST submission, 2010. Available at http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/bps/bps-spec.pdf.
6. CardHub. Number of credit cards and credit card holders, 2012. Available at http://www.cardhub.com/edu/number-of-credit-cards/.
7. V. T. Hoang, B. Morris, and P. Rogaway. An enciphering scheme based on a card shuffle. In R. Safavi-Naini and R. Canetti, editors, *CRYPTO*, volume 7417 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2012.
8. ISO/IEC 7812-1. Identification cards-identification of issuers-part 1: Numbering system, 2006.
9. M. Liskov, R. L. Rivest, and D. Wagner. Tweakable block ciphers. In M. Yung, editor, *CRYPTO*, volume 2442 of *Lecture Notes in Computer Science*, pages 31–46. Springer, 2002.
10. B. Morris, P. Rogaway, and T. Stegers. How to encipher messages on a small domain. In S. Halevi, editor, *CRYPTO*, volume 5677 of *Lecture Notes in Computer Science*, pages 286–302. Springer, 2009.
11. PCI Security Standards Council. Payment card industry data security standard version 1.2, 2008. Available at https://www.pcisecuritystandards.org/security_standards/pci_dss.shtml.
12. PCI Security Standards Council. Information supplement: PCI DSS tokenization guidelines, 2011. Available at https://www.pcisecuritystandards.org/documents/Tokenization_Guidelines_Info_Supplement.pdf.
13. T. Ristenpart and S. Yilek. The mix-and-cut shuffle: Small-domain encryption secure against n queries. In R. Canetti and J. A. Garay, editors, *CRYPTO (1)*, volume 8042 of *Lecture Notes in Computer Science*, pages 392–409. Springer, 2013.
14. M. J. B. Robshaw and O. Billet, editors. *New Stream Cipher Designs - The eSTREAM Finalists*, volume 4986 of *Lecture Notes in Computer Science*. Springer, 2008.
15. RSA White paper. Tokenization: What next after PCI, 2012. Available at http://www.emc.com/collateral/white-papers/h11918-wp-tokenization-rsa-dpm.pdf.
16. Securosis White Paper. Tokenization guidance: How to reduce pci compliance costs, 2011. Available at http://gateway.elavon.com/documents/Tokenization_Guidelines_White_Paper.pdf.
17. Securosis White Paper. Tokenization vs. encryption: Options for compliance, 2011. Available at https://securosis.com/research/publication/tokenization-vs.-encryption-options-for-compliance.
18. E. Stefanov and E. Shi. Fastprp: Fast pseudo-random permutations for small domains. *IACR Cryptology ePrint Archive*, 2012:254, 2012.
19. Voltage Security White paper. Payment security solution - processor edition, 2012. Available at http://www.voltage.com/wp-content/uploads/Voltage_White_Paper_SecureData_PaymentsProcessorEdition.pdf.

A Deferred Proofs

A.1 Proof of Theorem 1

We only prove the first claim in the theorem, as discussed earlier, the second claim directly follows from the first one. We construct a $\widetilde{\text{prp}}$ adversary $\mathcal{B}$ which runs an arbitrary adversary $\mathcal{A}$ who attacks TKR1. $\mathcal{B}$ being a $\widetilde{\text{prp}}$ adversary has access to an oracle $O(.,.)$, which is either the real tweakable permutation $\mathsf{FP}_k(.,.)$ for a

randomly chosen key k, or a random permutation chosen uniformly at random from the set of all tweak index permutations from $\mathcal{T}$ to $\mathcal{T}$. $\mathcal{B}$ with its oracle provides the environment to $\mathcal{A}$ and simulates the experiment EXP-IND-TKR$^{\mathcal{A}}_{\mathsf{TKR1}}$ as shown in Figure 9.

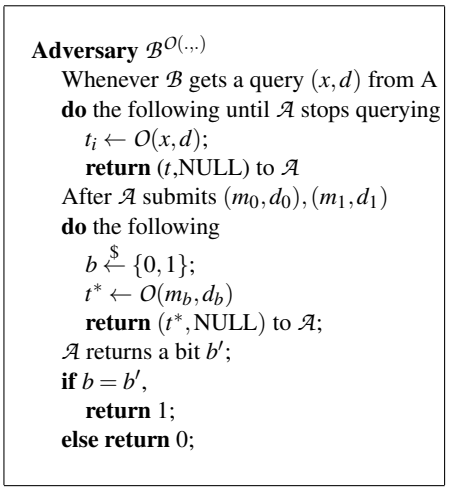

Fig. 9: Adversary $\mathcal{B}$ for the proof of Theorem 1

We assume without loss of generality that $\mathcal{A}$ does not repeat queries, as $\mathcal{A}$ knows that TKR1 is a deterministic scheme, hence it does not gain anything by repeating a query.

It is easy to see that if the oracle $O(.,.)$ of $\mathcal{B}$ is $\mathsf{FP}_k(.,.)$, then $\mathcal{B}^O$ provides the perfect environment for $\mathcal{A}$ as in EXP-IND-TKR$^{\mathcal{A}}_{\mathsf{TKR1}}$. Hence,

$$\Pr[k \xleftarrow{\$} \mathcal{K} : \mathcal{B}^{\mathsf{FP}_k(.,.)} \Rightarrow 1] = \Pr[\text{EXP-IND-TKR}^{\mathcal{A}}_{\mathsf{TKR1}} \Rightarrow 1]. \tag{3}$$

Also,

$$\Pr[\pi \xleftarrow{\$} \mathbf{Perm}^{\mathcal{D}}(\mathcal{T}) : \mathcal{B}^{\pi(.,.)} \Rightarrow 1] \leq \frac{1}{2}, \tag{4}$$

as, when $O(.,.)$ is a uniform random tweakable permutation on $\mathcal{T}$, for each of its queries $\mathcal{A}$ gets uniform random elements in $\mathcal{T}$, thus b' which $\mathcal{A}$ outputs is independent of b which is selected by $\mathcal{B}$.

Hence from equations (3) and (4), we have

$$\mathbf{Adv}^{\widetilde{\text{prp}}}_{\mathsf{FP}}(\mathcal{B}) \geq \Pr[\text{EXP-IND-TKR}^{\mathcal{A}}_{\mathsf{TKR1}} \Rightarrow 1] - \frac{1}{2},$$

and hence

$$\mathbf{Adv}^{\text{ind-tkr}}_{\mathsf{TKR1}}(\mathcal{A}) \leq \mathbf{Adv}^{\widetilde{\text{prp}}}_{\mathsf{FP}}(\mathcal{B}),$$

as desired. □

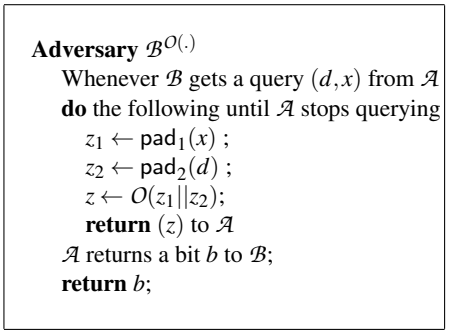

Fig. 10: Adversary $\mathcal{B}$ for the proof of Proposition 2.

A.2 Proof of Proposition 2

To prove this proposition, we construct a prf adversary $\mathcal{B}$ (shown in Fig. 10) which runs an arbitrary adversary $\mathcal{A}$ who attacks the encryption scheme **E1** in the det-cpa sense. $\mathcal{B}$ being a prf adversary has access to an oracle O which can be either be the block cipher E_k or a function ρ, chosen uniformly at random from $\mathsf{Func}(n)$.

We can easily see that if the oracle of $\mathcal{B}$ is the block cipher E_k then

$$\Pr[k \xleftarrow{\$} \mathbb{K} : \mathcal{B}^{E_k(\cdot)} \Rightarrow 1] = \Pr[k \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{\mathbf{E1}(\cdot,\cdot)} \Rightarrow 1]. \tag{5}$$

As $\mathcal{A}$ never repeats a query, so if the oracle of $\mathcal{B}$ is a random function ρ, then for each query $\mathcal{A}$ gets a uniform random n bit string as a response. Thus,

$$\Pr[\rho \xleftarrow{\$} \mathsf{Func}(n) : \mathcal{B}^{\rho(\cdot)} \Rightarrow 1] = \Pr[\mathcal{A}^{\$(\cdot,\cdot)} \Rightarrow 1] \tag{6}$$

Thus from the equations above, and the definition of the det-cpa advantage of $\mathcal{A}$ and the prf advantage of $\mathcal{B}$, we obtain

$$\mathbf{Adv}^{\text{det-cpa}}_{\mathbf{E1}}(\mathcal{A}) = \mathbf{Adv}^{\text{prf}}_{E}(\mathcal{B}).$$

□

A.3 Proof of Proposition 3

As in the proof of Proposition 3, we construct a prf adversary $\mathcal{B}$ (shown in Fig. 11) which runs an arbitrary adversary $\mathcal{A}$ who attacks the encryption scheme **E2**. Adversary $\mathcal{B}$ has access to an oracle O which can be either a secure block cipher E_k or a pseudorandom function ρ, chosen uniformly at random from $\mathsf{Func}(n)$.

We can easily see that if the oracle of $\mathcal{B}$ is the block cipher E_k then

$$\Pr[k \xleftarrow{\$} \mathbb{K} : \mathcal{B}^{E_k(\cdot)} \Rightarrow 1] = \Pr[k \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{\mathbf{E2}(\cdot,\cdot)} \Rightarrow 1] \tag{7}$$

To analyze the situation when the oracle of $\mathcal{B}$ is a random function, we consider the game **G0** shown in Figure 12. The game **G0** describes a function **Choose**-$\rho()$, which acts as a random function. It returns uniform random strings in $\{0,1\}^n$ when it is invoked, but it returns the same string if invoked twice on the

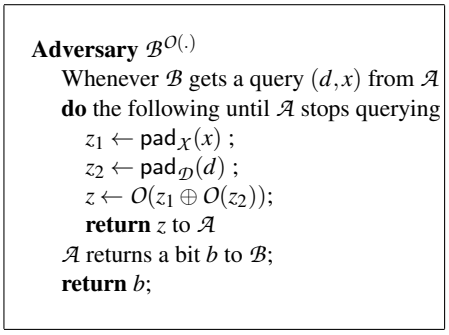

Fig. 11: Adversary $\mathcal{B}$ for the proof of Proposition 3.

same input. It does this by maintaining a table ρ of outputs that it has already returned. Additionally in the set DOM, it maintains the points on which it has been queried. The function sets the bad flag to true if it is queried twice on the same input.

As **Choose**-ρ acts like a random function, hence it is immediate that

$$\Pr[\rho \xleftarrow{\$} \mathsf{Func}(n) : \mathcal{B}^{\rho(\cdot)} \Rightarrow 1] = \Pr[\mathcal{A}^{G0} \Rightarrow 1] \tag{8}$$

Now, we do a small change in game **G0**, i.e., we remove the boxed entry in the function **Choose**-ρ, we call this changed game as **G1**. Notice that games **G1** and **G0** are identical until the flag bad is set to true, hence we have

$$\Pr[\mathcal{A}^{G0} \Rightarrow 1] - \Pr[\mathcal{A}^{G1} \Rightarrow 1] \leq \Pr[\mathcal{A}^{G1} \text{ sets } \mathsf{bad}] \tag{9}$$

Also in game **G1**, the function **Choose**-ρ, returns random strings for any input it gets, thus $\mathcal{A}$ when interacts with **G1** gets random strings in $\{0,1\}^n$ in response to its queries. Hence,

$$\Pr[\mathcal{A}^{\$(\cdot,\cdot)} \Rightarrow 1] = \Pr[\mathcal{A}^{G1} \Rightarrow 1]. \tag{10}$$

Now, we do some small syntactic changes in the game **G1** to obtain the game **G2**, also shown in Figure 12. Game **G2** is only syntactically different from **G1**. In **G2** random strings are returned immediately as a response to a query of $\mathcal{A}$, and later in the finalization phase appropriate values are inserted in the multiset Dom, note as Dom is a multiset hence there can be several instances of the same element present here.

As, there is no way that $\mathcal{A}$ can distinguish between **G1** and **G2**, hence

$$\Pr[\mathcal{A}^{G1} \Rightarrow 1] = \Pr[\mathcal{A}^{G2} \Rightarrow 1], \tag{11}$$

also

$$\Pr[\mathcal{A}^{G1} \text{ sets } \mathsf{bad}] = \Pr[\mathcal{A}^{G2} \text{ sets } \mathsf{bad}]. \tag{12}$$

Thus, using equations (8), (9), (10), (11) and (12) we get

$$\begin{aligned} \Pr[\rho \xleftarrow{\$} \mathsf{Func}(n) : \mathcal{B}^{\rho(\cdot)} \Rightarrow 1] &= \Pr[\mathcal{A}^{G0} \Rightarrow 1] \\ &\leq \Pr[\mathcal{A}^{G1} \Rightarrow 1] + \Pr[\mathcal{A}^{G1} \text{ sets } \mathsf{bad}] \\ &\leq \Pr[\mathcal{A}^{G2} \Rightarrow 1] + \Pr[\mathcal{A}^{G2} \text{ sets } \mathsf{bad}] \\ &\leq \Pr[\mathcal{A}^{\$(\cdot,\cdot)} \Rightarrow 1] + \Pr[\mathcal{A}^{G2} \text{ sets } \mathsf{bad}] \end{aligned} \tag{13}$$

Game $\boxed{\textbf{G0}}$, **G1**

function Choose-$\rho(X)$
  $Y \xleftarrow{\$} \{0,1\}^n$;
  if $X \in$ DOM **then**
    bad $\leftarrow$ true
    $\boxed{Y \leftarrow \rho[X]}$
  else
    $\rho[X] \leftarrow Y$
    Dom $\leftarrow$ Dom $\cup \{X\}$
  end if
  return Y
Initialization
  bad $\leftarrow$ false;
  Dom $= \emptyset$;
Query Phase
For a query $(d^{(i)}, x^{(i)})$ of $\mathcal{A}$ do the following
  $z_1^{(i)} \leftarrow \mathsf{pad}_{\mathcal{X}}(x^{(i)})$;
  $z_2^{(i)} \leftarrow \mathsf{pad}_{\mathcal{D}}(d^{(i)})$;
  if $z_2^{(i)} = z_2^{(j)}$ for some $j < i$ **then** $\mu^{(i)} \leftarrow \mu^{(j)}$
  else $\mu^{(i)} \leftarrow$ **Choose-**$\rho(z_2^{(i)})$
  end if
  $\lambda^{(i)} \leftarrow z_1^{(i)} \oplus \mu^{(i)}$
  $z^{(i)} \leftarrow$ **Choose-**$\rho(\lambda^{(i)})$
  return z

Game **G2**

Query Phase
For a query $(d^{(i)}, x^{(i)})$ of $\mathcal{A}$, do the following
  $z^{(i)} \xleftarrow{\$} \{0,1\}^n$
  return $z^{(i)}$

Finalization
for $i \rightarrow 1$ to q
  $z_1^{(i)} \leftarrow \mathsf{pad}_{\mathcal{X}}(x^{(i)})$;
  $z_2^{(i)} \leftarrow \mathsf{pad}_{\mathcal{D}}(d^{(i)})$;
  if $z_2^{(i)} = z_2^{(j)}$ for $j < i$ **then**
    $\mu^{(i)} \leftarrow \mu^{(j)}$
  else
    $\mu^{(i)} \xleftarrow{\$} \{0,1\}^n$
    Dom $\leftarrow$ Dom $\cup \{z_2^{(i)}\}$
  end if
  $\lambda^{(i)} \leftarrow z_1^{(i)} \oplus \mu^{(i)}$
  Dom $\leftarrow$ Dom $\cup \{\lambda^{(i)}\}$
endfor
  if there is a collision in Dom then
    bad $\leftarrow$ true

Fig. 12: Games **G0, G1, G2** used for the proof of Proposition 3.

Let COLLD be the event that there is a collision in the multiset Dom in game **G2**, then from the description of game **G2**, we have

$$\Pr[\mathcal{A}^{G2} \text{ sets bad}] = \Pr[\mathsf{COLLD}]$$

Now we concentrate on finding an upper bound for $\Pr[\mathsf{COLLD}]$. The elements present in Dom are d's and λ's. Let $\mathsf{Dom} = Q_d \cup Q_\lambda$, where $Q_d \subseteq \{d^{(i)} : 1 \le i \le q\}$, and $Q_\lambda = \{\lambda^{(i)} = z^{(i)} \oplus \mu^{(i)} | 1 \le i \le q\}$.

Note, that the way the game **G2** is designed, all elements in Q_d are distinct, thus there can be no collision among two elements in Q_d. Additionally we claim the following

Claim 1 *For* $1 \le i, j \le q$, $i \ne j$, $\Pr[\lambda^{(i)} = \lambda^{(j)}] \le 1/2^n$.

Proof. We have two cases to consider:
Case 1. If $d^{(i)} = d^{(j)}$, then $x^{(i)} \ne x^{(j)}$, as $\mathcal{A}$ does not repeat any query. This makes $z^{(i)} \ne z^{(j)}$. According to the game **G2**, if $d^{(i)} = d^{(j)}$, then $\mu^{(i)} = \mu^{(j)}$. Thus we have $\lambda^{(i)} \ne \lambda^{(j)}$. Thus, making $\Pr[\lambda^{(i)} = \lambda^{(j)}] = 0$.
Case 2. If $d^{(i)} \ne d^{(j)}$, then $\mu^{(i)}$ and $\mu^{(j)}$ are uniform and independent random elements in $\{0,1\}^n$, thus making

$$\Pr[\lambda^{(i)} = \lambda^{(j)}] = \Pr[z_1^{(i)} \oplus \mu^{(i)} = z_1^{(j)} \oplus \mu^{(j)}] = \frac{1}{2^n}.$$

Claim 2 *For any $d \in Q_d$ and any $\lambda \in Q_\lambda$, $\Pr[\lambda = d] \leq 1/2^n$.*

Proof. Any $\lambda \in Q_\lambda$ is a uniform random string in $\{0,1\}^n$, and is independent of any $d \in Q_d$.

Now, as $\#Q_d \leq q$ and $\#Q_\lambda = q$, using Claims 1, 2 and the union bound, we have

$$\Pr[\mathsf{COLLD}] \leq \frac{1}{2^n}\binom{q}{2} + \frac{q^2}{2^n} < \frac{2q^2}{2^n}.$$

Now, using the definition of det-cpa advantage of $\mathcal{A}$ and equations (7) and (13), we have the proposition. □

A.4 Proof of Theorem 3

Note that the token generation algorithm for both TKR2 and TKR2a are the same, the only difference between the two procedures is the structure and content of the card-vault. Hence the proof of security in IND-TKR sense for both TKR2 and TKR2a are same, as in case of IND-TKR security the adversary does not have access to the contents of the card-vault.

The structure of the proof is same as the proof of Theorem 1. We assume an arbitrary adversary $\mathcal{A}$ which attacks TKR2 in IND-TKR sense, and we construct a rnd adversary $\mathcal{B}$ which attacks $\mathsf{RN}^{\mathcal{T}}[k]$ using $\mathcal{A}$.

$\mathcal{B}$ has an oracle O, which is either $\mathsf{RN}^{\mathcal{T}}[k]$ for a random key, or $\$^{\mathcal{T}}()$, which on each invocation returns a random element in $\mathcal{T}$.

$\mathcal{B}$ responds to queries of $\mathcal{A}$ as follows. First $\mathcal{B}$ initiates with an empty card-vault and then perform the query phase, which in fact is the procedure $\mathsf{TKR2}_k$ in Figure 5. Only when a call to $\mathsf{RN}^{\mathcal{T}}[k]()$ is required, it is replaced by a call to its oracle O. After $\mathcal{A}$ stops querying and outputs the challenge pair $(m_0,d_0),(m_1,d_1)$, $\mathcal{B}$ selects a bit b uniformly at random from $\{0,1\}$ and provides $\mathcal{A}$ with t computed by following $\mathsf{TKR2}_k()$ (the call to $\mathsf{RN}^{\mathcal{T}}[k]()$ replaced by a call to O). Finally $\mathcal{A}$ outputs a bit b', and if $b = b'$, then $\mathcal{B}$ outputs 1 else outputs a 0. Note that the challenge pair $(m_0,d_0),(m_1,d_1)$, is different from any previous query of $\mathcal{A}$.

From the above description it is clear that if the oracle $O(.,.)$ of $\mathcal{B}$ is $\mathsf{RN}^{\mathcal{T}}[k]()$, then $\mathcal{B}$ is performing experiment $\text{EXP-IND-TKR}^{\mathcal{A}}_{\mathsf{TKR2}}$. Hence

$$\Pr[k \xleftarrow{\$} \mathcal{K} : \mathcal{B}^{\mathsf{RN}^{\mathcal{T}}[k]()} \Rightarrow 1] = \Pr[\text{EXP-IND-TKR}^{\mathcal{A}}_{\mathsf{TKR2}} \Rightarrow 1]. \tag{14}$$

Otherwise, i.e. if the oracle $O(.,.)$ of $\mathcal{B}$ is $\$^{\mathcal{T}}()$ then,

$$\Pr[\mathcal{B}^{\$^{\mathcal{T}}()} \Rightarrow 1] \leq \frac{1}{2}. \tag{15}$$

As in this case the output that $\mathcal{B}$ provides to $\mathcal{A}$ is independent of $(m_0,d_0),(m_1,d_1)$.

From equations (14), (15) we have,

$$\mathbf{Adv}^{\text{rnd}}_{\mathsf{RN}}(\mathcal{B}) \geq \Pr[\text{EXP-IND-TKR}^{\mathcal{A}}_{\mathsf{TKR2}} \Rightarrow 1] - \frac{1}{2},$$

and from the definition of IND-TKR advantage of $\mathcal{A}$ it follows

$$\mathbf{Adv}^{\text{ind-tkr}}_{\mathsf{TKR2}}(\mathcal{A}) \leq \mathbf{Adv}^{\text{rnd}}_{\mathsf{RN}}(\mathcal{B}).$$

□

A.5 Proof of Theorem 4

For this proof we use the sequence of games. The three games $\mathbf{EXP}_0^{\mathcal{A}}$, $\mathbf{EXP}_1^{\mathcal{A}}$ and $\mathbf{EXP}_2^{\mathcal{A}}$ are described in Figure 13. Each game depicts the interaction of an IND-TKR-CV adversary with a tokenization procedure. In all the three games we assume that the adversary $\mathcal{A}$ does not repeat a query in the query phase, and the queries presented in the challenge phase are also distinct from the queries made in the query phase. Also, to keep things simple in terms of notations, without loss of generality we assume that the ciphertext space $\mathbb{C}$ of the encryption algorithm $\mathbf{E}$ contains strings of length s. The proof can be made to work without this restriction. We describe the three different games briefly next:

1. In game $\mathbf{EXP}_0^{\mathcal{A}}$, $\mathcal{A}$ interacts with $\mathsf{TKR2a}$, instantiated by $\mathsf{RN}^T[k_2]()$ and $\mathbf{E}_{k_1}(\cdot,\cdot)$, where k_1 and k_2 are chosen uniformly at random from the respective key spaces $\mathcal{K}_1$ and $\mathcal{K}_2$. The game is designed with the assumption that, $\mathcal{A}$ does not repeat a query.
2. Game $\mathbf{EXP}_1^{\mathcal{A}}$ is almost same as the game $\mathbf{EXP}_0^{\mathcal{A}}$. The differences are as follows:
   - Here the encryption scheme $\mathbf{E}_{k_1}(\cdot,\cdot)$, is no more used. Instead, each call to $\mathbf{E}_{k_1}(\cdot,\cdot)$ is responded by a random string from $\mathbb{C}$. To maintain the same behaviour of $\mathbf{E}_{k_1}$, a set Ran_1 is maintained to keep track of the values already returned as output, and it is ensured that the same value is not returned for two different inputs.
   - In the game $\mathbf{EXP}_0^{\mathcal{A}}$, in lines 11 to 14 and 53 to 56 it is ensured that a distinct token is t returned for each distinct (x,d). This is done by a search in the card-vault (see lines 14 and 56), as the card-vault contains encryption of the token t with associated data d. As in the game $\mathbf{EXP}_1^{\mathcal{A}}$, a real encryption scheme is not used, so this search is not possible. Hence a set Tok is maintained which contains pairs of tokens and associated data (t,d) and the uniqueness of tokens is ensured using this set Tok.
3. Game $\mathbf{EXP}_2^{\mathcal{A}}$ is obtained from game $\mathbf{EXP}_1^{\mathcal{A}}$ by replacing $\mathsf{RN}^{\mathcal{T}}[k_2]()$ by a procedure which on each invocation returns a random element in $\mathcal{T}$. This game also used the sets Ran_1 and Tok to ensure injectivity and the uniqueness of the tokens.

It is easy to see that $\mathbf{EXP}_0^{\mathcal{A}}$ is a restatement of the experiment Exp-IND-TKR-CV$^{\mathcal{A}}$ in Figure 2. Hence,

$$\Pr[\text{Exp-IND-TKR-CV}^{\mathcal{A}} \Rightarrow 1] = \Pr[\mathbf{EXP}_0^{\mathcal{A}} \Rightarrow 1]. \tag{16}$$

Also we make the following claims:

Claim 3 *There exists a det-cpa adversary $\mathcal{B}$ for $\mathbf{E}$ such that*

$$\Pr[\mathbf{EXP}_0^{\mathcal{A}} \Rightarrow 1] - \Pr[\mathbf{EXP}_1^{\mathcal{A}} \Rightarrow 1] \leq \mathbf{Adv}_{\mathbf{E}}^{\text{det-cpa}}(\mathcal{B}) + \frac{(q+1)^2}{2^s}$$

Proof. To prove this claim we construct a det-cpa adversary $\mathcal{B}$ which has access to an oracle O. This oracle is either the encryption scheme $\mathbf{E}_{k_1}$ for a random key k_1 or $\$(\cdot,\cdot)$ which on input (x,d) returns random strings of length s. $\mathcal{B}$ has the objective of distinguishing between these two scenarios. $\mathcal{B}$ runs $\mathcal{A}$ in the following way. First $\mathcal{B}$ initiates with an empty card-vault and selects a random key k_2 from $\mathcal{K}_2$, and also initializes a multi-set Dom to empty. Then, it answers queries of $\mathcal{A}$ according to the procedure $\mathsf{TKR2a}$ (shown in Figure 6). To answer the queries, whenever a call to the encryption scheme $\mathbf{E}_{k_1}$ is required, it is replaced by a call to its oracle O. $\mathcal{B}$ also stores each output it gets from its oracle O in the set Dom. Note, as $\mathcal{A}$ does not repeat any query, hence all queries made by $\mathcal{B}$ to its oracle is distinct. After $\mathcal{A}$ stops querying and outputs a challenge pair $(x_0,d_0),(x_1,d_1)$, $\mathcal{B}$ selects a bit uniformly at random from $\{0,1\}$ and provides $\mathcal{A}$ with the

Game $\mathbf{EXP}_0^{\mathcal{A}}$

Initialization:

01. $\mathsf{CV} \leftarrow$ NULL;
02. $k_1 \xleftarrow{\$} \mathcal{K}_1$;
03. $k_2 \xleftarrow{\$} \mathcal{K}_2$;

Query Phase

Respond to a query (x,d)
by $\mathcal{A}$ as follows

10. $z \leftarrow \mathbf{E}_{k_1}(x,d)$;
11. **do**
12. $\quad t \leftarrow \mathsf{RN}^{\mathcal{T}}[k_2]()$;
13. $\quad t' \leftarrow \mathbf{E}_{k_1}(d, \mathsf{b}||t)$;
14. **while** $\mathsf{SrchCV}(1,t') \neq \emptyset$;
15. $c \leftarrow (t',z)$;
16. $\mathsf{InsertCV}(c)$;
17. **return** (t,c) to $\mathcal{A}$

Challenge Phase

After $\mathcal{A}$ submits $(x_0,d_0),(x_1,d_1)$
do the following:

51. $b \xleftarrow{\$} \{0,1\}$;
52. $z \leftarrow \mathbf{E}_{k_1}(x_b,d_b)$;
53. **do**
54. $\quad t \leftarrow \mathsf{RN}^{\mathcal{T}}[k_2]()$;
55. $\quad t' \leftarrow \mathbf{E}_{k_1}(d, \mathsf{b}||t)$;
56. **while** $\mathsf{SrchCV}(1,t') \neq \emptyset$;
57. $c \leftarrow (t',z)$;
58. **return** (t,c) to $\mathcal{A}$

Finalization Phase

After $\mathcal{A}$ outputs the bit b'
do the following:

80. **if** $b = b'$ **output** 1
81. **else output** 0

Game $\mathbf{EXP}_1^{\mathcal{A}}$

Initialization:

01. $\mathsf{CV} \leftarrow$ NULL;
02. $k_2 \leftarrow \mathcal{K}_2$;
03. $\mathsf{Ran}_1 \leftarrow \emptyset$
04. $\mathsf{Tok} \leftarrow \emptyset$

Query Phase

Respond to a query (x,d)
by $\mathcal{A}$ as follows

10. $z \xleftarrow{\$} \mathbb{C} \setminus \mathsf{Ran}_1$;
11. $\mathsf{Ran}_1 \leftarrow \mathsf{Ran}_1 \cup \{z\}$;
12. **do**, $t \leftarrow \mathsf{RN}^{\mathcal{T}}[k_2]()$;
13. **while** $\mathsf{Tok} \cap \{(t,d)\} \neq \emptyset$;
14. $\mathsf{Tok} \leftarrow \mathsf{Tok} \cup \{(t,d)\}$;
15. $t' \xleftarrow{\$} \mathbb{C} \setminus \mathsf{Ran}_1$;
16. $\mathsf{Ran}_1 \leftarrow \mathsf{Ran}_1 \cup \{t'\}$;
17. $c \leftarrow (t',z)$;
18. $\mathsf{InsertCV}(c)$;
19. **return** (t,c) to $\mathcal{A}$

Challenge Phase

After $\mathcal{A}$ submits $(x_0,d_0),(x_1,d_1)$
do the following:

51. $b \xleftarrow{\$} \{0,1\}$;
52. $z \xleftarrow{\$} \mathbb{C} \setminus \mathsf{Ran}_1$;
53. $\mathsf{Ran}_1 \leftarrow \mathsf{Ran}_1 \cup \{z\}$;
54. **do** $t \leftarrow \mathsf{RN}^{\mathcal{T}}[k_2]()$;
55. **while** $\mathsf{Tok} \cap \{(t,d_b)\} \neq \emptyset$;
56. $\mathsf{Tok} \leftarrow \mathsf{Tok} \cup \{(t,d_b)\}$;
57. $t' \xleftarrow{\$} \mathbb{C} \setminus \mathsf{Ran}_1$;
58. $\mathsf{Ran}_1 \leftarrow \mathsf{Ran}_1 \cup \{t'\}$;
59. $c \leftarrow (t',z)$;
60. **return** (t,c) to $\mathcal{A}$

Finalization Phase

After $\mathcal{A}$ outputs the bit b'
do the following:

80. **if** $b = b'$ **output** 1
81. **else output** 0

Game $\mathbf{EXP}_2^{\mathcal{A}}$

Initialization:

01. $\mathsf{CV} \leftarrow$ NULL;
02. $\mathsf{Ran}_1 \leftarrow \emptyset$
03. $\mathsf{Tok} \leftarrow \emptyset$

Query Phase

Respond to a query (x,d)
by $\mathcal{A}$ as follows:

10. $z \xleftarrow{\$} \mathbb{C} \setminus \mathsf{Ran}_1$;
11. $\mathsf{Ran}_1 \leftarrow \mathsf{Ran}_1 \cup \{z\}$;
12. **do** $t \xleftarrow{\$} \mathcal{T}$;
13. **while** $\mathsf{Tok} \cap \{(t,d)\} \neq \emptyset$;
14. $\mathsf{Tok} \leftarrow \mathsf{Tok} \cup \{(t,d)\}$;
15. $t' \xleftarrow{\$} \mathbb{C} \setminus \mathsf{Ran}_1$;
16. $\mathsf{Ran}_1 \leftarrow \mathsf{Ran}_1 \cup \{t'\}$;
17. $c \leftarrow (t',z)$;
18. $\mathsf{InsertCV}(c)$;
19. **return** (t,c) to $\mathcal{A}$

Challenge Phase

After $\mathcal{A}$ submits $(x_0,d_0),(x_1,d_1)$
do the following:

51. $b \xleftarrow{\$} \{0,1\}$;
52. $z \xleftarrow{\$} \mathbb{C} \setminus \mathsf{Ran}_1$;
53. $\mathsf{Ran}_1 \leftarrow \mathsf{Ran}_1 \cup \{z\}$;
54. **do**, $t \xleftarrow{\$} \mathcal{T}$;
55. **while** $\mathsf{Tok} \cap \{(t,d_b)\} \neq \emptyset$;
56. $\mathsf{Tok} \leftarrow \mathsf{Tok} \cup \{(t,d_b)\}$;
57. $t' \xleftarrow{\$} \mathbb{C} \setminus \mathsf{Ran}_1$;
58. $\mathsf{Ran}_1 \leftarrow \mathsf{Ran}_1 \cup \{t'\}$;
59. $c \leftarrow (t',z)$;
60. **return** (t,c) to $\mathcal{A}$

Finalization Phase

After $\mathcal{A}$ outputs the bit b'
do the following:

80. **if** $b = b'$ **output** 1
81. **else output** 0

Fig. 13: The three games used to prove Theorem 4

pair (t,c). For responding to $\mathcal{A}$'s challenge, $\mathcal{B}$ makes another call to O and the output of O for this call is also inserted in Dom. Finally $\mathcal{A}$ outputs a bit b'. Now, $\mathcal{B}$ checks if there is a collision in Dom, i.e., if O ever returned two same values for two distinct queries. If there is a collision in Dom, then $\mathcal{B}$ outputs 0. On the other hand, if there is no collision in Dom and $b = b'$ then $\mathcal{B}$ outputs 1, otherwise it outputs a 0.

From the description above, we can easily see that if the oracle of $\mathcal{B}$ is the encryption scheme $\mathbf{E}_{k_1}(\cdot,\cdot)$, then there is never a collision in Dom as $\mathbf{E}_{k_2}(\cdot,\cdot)$ is injective, and in this scenario $\mathcal{B}$ is providing the exact environment of the game $\mathbf{EXP}_0^{\mathcal{A}}$, i.e.

$$\Pr[k_1 \xleftarrow{\$} \mathcal{K}_1 : \mathcal{B}^{\mathcal{E}_K(\cdot,\cdot)} \Rightarrow 1] \leq \Pr[\mathbf{EXP}_0^{\mathcal{A}} \Rightarrow 1]. \tag{17}$$

On the other hand, if the oracle of $\mathcal{B}$ is $\$(\cdot,\cdot)$, then $\mathcal{B}$ is providing the environment of $\mathbf{EXP}_1^{\mathcal{A}}$, given that there is no collision in Dom. If COLL be the event that there is a collision in Dom, then we have,

$$\begin{aligned}
\Pr[\mathcal{B}^{\$(\cdot,\cdot)} \Rightarrow 0] &= \Pr[(\mathcal{B}^{\$(\cdot,\cdot)} \Rightarrow 0) \wedge (\mathsf{COLL} \vee \overline{\mathsf{COLL}})] \\
&= \Pr[(\mathcal{B}^{\$(\cdot,\cdot)} \Rightarrow 0) \wedge \mathsf{COLL}] + \Pr[(\mathcal{B}^{\$(\cdot,\cdot)} \Rightarrow 0) \wedge \overline{\mathsf{COLL}})] \\
&= \Pr[(\mathcal{B}^{\$(\cdot,\cdot)} \Rightarrow 0) | \mathsf{COLL}] \Pr[\mathsf{COLL}] + \Pr[(\mathcal{B}^{\$(\cdot,\cdot)} \Rightarrow 0) | \overline{\mathsf{COLL}}] \Pr[\overline{\mathsf{COLL}}] \\
&\geq \Pr[\mathbf{EXP}_1^{\mathcal{A}} \Rightarrow 0](1 - \Pr[\mathsf{COLL}]).
\end{aligned}$$

Thus

$$\begin{aligned}
\Pr[\mathcal{B}^{\$(\cdot,\cdot)} \Rightarrow 1] &\leq \Pr[\mathbf{EXP}_1^{\mathcal{A}} \Rightarrow 1] + \Pr[\mathbf{EXP}_1^{\mathcal{A}} \Rightarrow 0] \Pr[\mathsf{COLL}] \\
&\leq \Pr[\mathbf{EXP}_1^{\mathcal{A}} \Rightarrow 1] + \Pr[\mathsf{COLL}]
\end{aligned} \tag{18}$$

Now from equations (17) and (18), and the definition of det-cpa advantage of $\mathcal{B}$, we have

$$\mathbf{Adv}_{\mathbf{E}}^{\text{det-cpa}}(\mathcal{B}) \geq \Pr[\mathbf{EXP}_0^{\mathcal{A}} \Rightarrow 1] - \Pr[\mathbf{EXP}_1^{\mathcal{A}} \Rightarrow 1] - \Pr[\mathsf{COLL}].$$

As, $\mathcal{A}$ asks q queries in the query phase, hence Dom has $q+1$ elements in it, and each element is a uniform random element in $\mathbb{C}$, and each element in $\mathbb{C}$ is s bits long. Hence,

$$\Pr[\mathsf{COLL}] = \binom{q+1}{2} \frac{1}{2^s} \leq \frac{(q+1)^2}{2^{s+1}}.$$

This completes the proof of the claim.

Claim 4 *There exists a* rnd *adversary* $\mathcal{B}'$ *such that*

$$\Pr[\mathbf{EXP}_1^{\mathcal{A}} \Rightarrow 1] - \Pr[\mathbf{EXP}_2^{\mathcal{A}} \Rightarrow 1] \leq \mathbf{Adv}_{\mathsf{RN}^{\mathcal{T}}}^{\text{rnd}}(\mathcal{B}')$$

Proof. The proof of this claim is an easy reduction. Again we have an adversary $\mathcal{A}$ attacking $\mathsf{TKR2a}$ and we must construct a rnd adversary $\mathcal{B}'$, which runs $\mathcal{A}$. $\mathcal{B}'$ has access to an oracle O, that could be either $\mathsf{RN}^T[k_2]()$ or $\$^{\mathcal{T}}$, which on each invocation it returns a random element in $\mathcal{T}$. As in Claim 3, adversary $\mathcal{B}'$ do an initialization and a query phase, but now when a call to $\mathsf{RN}^T[k]()$ is required, it is substituted by a call to the oracle O. Now we can see that

$$\Pr[k \xleftarrow{\$} \mathcal{K} : \mathcal{B}'^{\mathsf{RN}^{\mathcal{T}}[k]()} \Rightarrow 1] = \Pr[\mathbf{EXP}_1^{\mathcal{A}} \Rightarrow 1] \tag{19}$$

in the case that the oracle of $\mathcal{B}$ is $\mathsf{RN}^T[k]()$, otherwise i.e., if O is $\$^{\mathcal{T}}$ then

$$\Pr[\mathcal{B}'^{\$^{\mathcal{T}}()} \Rightarrow 1] \leq \Pr[\mathbf{EXP}_2^{\mathcal{A}} \Rightarrow 1] \tag{20}$$

Again from equations (19) and (20), the claim follows.

Claim 5 *For any arbitrary adversary* $\mathcal{A}$

$$\Pr[\mathbf{EXP}_2^{\mathcal{A}} \Rightarrow 1] = \frac{1}{2}$$

Proof. In game $\mathbf{EXP}_2^{\mathcal{A}}$, in the query phase $\mathcal{A}$ receives q tuples (t,c) where t and c are distinct random elements in $\mathcal{T}$ and $\mathbb{C}$, respectively. Finally in the challenge phase it receives (t,c) which is independent of $(x_0,d_0),(x_1,d_1)$. Hence, $\mathcal{A}$ cannot only guess the bit b with probability more than $\frac{1}{2}$.

Thus, from Claims 3, 4,

$$\Pr[\mathbf{EXP}_0^{\mathcal{A}} \Rightarrow 1] - \Pr[\mathbf{EXP}_2^{\mathcal{A}} \Rightarrow 1] \leq \mathbf{Adv}_{\mathbf{E}}^{\text{det-cpa}}(\mathcal{B}) + \mathbf{Adv}_{\mathsf{RN}^{\mathcal{T}}}^{\text{rnd}}(\mathcal{B}') + \frac{(q+1)^2}{2^{s+1}} \tag{21}$$

Using equation (16) and claim 5,

$$\Pr[\text{Exp-IND-TKR-CV}^{\mathcal{A}} \Rightarrow 1] - \frac{1}{2} \leq \mathbf{Adv}_{\mathbf{E}}^{\text{det-cpa}}(\mathcal{B}) + \mathbf{Adv}_{\mathsf{RN}^{\mathcal{T}}}^{\text{rnd}}(\mathcal{B}') + \frac{(q+1)^2}{2^{s+1}}. \tag{22}$$

Finally, we have

$$\mathbf{Adv}_{\Psi}^{\text{ind-tkr-cv}}(\mathcal{A}) \leq \mathbf{Adv}_{\mathsf{RN}}^{\text{rnd}}(\mathcal{B}) + \mathbf{Adv}_{\mathbf{E}}^{\text{det-cpa}}(\mathcal{B}') + \frac{(q+1)^2}{2^{s+1}},$$

as desired. □