Modifying an Enciphering Scheme after Deployment*

Paul Grubbs[1], Thomas Ristenpart[1], and Yuval Yarom[2]
[1] Cornell Tech [2] University of Adelaide and Data61, CSIRO

Abstract

Assume that a symmetric encryption scheme has been deployed and used with a secret key. We later must change the encryption scheme in a way that preserves the ability to decrypt (a subset of) previously encrypted plaintexts. Frequent real-world examples are migrating from a token-based encryption system for credit-card numbers to a format-preserving encryption (FPE) scheme, or extending the message space of an already deployed FPE. The ciphertexts may be stored in systems for which it is not easy or not efficient to retrieve them (to re-encrypt the plaintext under the new scheme).

We introduce methods for functionality-preserving modifications to encryption, focusing particularly on deterministic, length-preserving ciphers such as those used to perform format-preserving encryption. We provide a new technique, that we refer to as the Zig-Zag construction, that allows one to combine two ciphers using different domains in a way that results in a secure cipher on one domain. We explore its use in the two settings above, replacing token-based systems and extending message spaces. We develop appropriate security goals and prove security relative to them assuming the underlying ciphers are themselves secure as strong pseudorandom permutations.

1 Introduction

We explore the ability to modify a deployed symmetric encryption scheme in a way that preserves some of its previous input-output mappings. This may prove useful in a variety of settings, but we are motivated and will focus on addressing two specific ones that arise in the increasing deployment of format-preserving encryption (FPE) schemes.

Modifying deployed FPE schemes. In a variety of settings conventional symmetric encryption is difficult or impossible to utilize, due to unfortunate constraints imposed by legacy software systems. A common problem is that encryption produces ciphertexts whose format are ruled out by restrictive application programming interfaces (APIs) and/or pre-existing database schema. This problem prevented, for example, encryption of credit-card numbers (CCNs) in a variety of settings. Format-preserving encryption (hereafter FPE) is a technique aimed at such problems, allowing one to encrypt a plaintext item of some format to a ciphertext of the same format (16 digit CCN). It has seen wide academic study [3, 7, 11, 17, 18, 20] as well as widespread use in industry [5, 12, 13, 22].

Before the advent of strong FPE schemes, companies often instead used what are called *tokenization systems* to solve the format-constrained ciphertext problem. One generates a random *token* using generic techniques for creating random strings with a certain format, i.e., sampling a token C uniformly from some set $\mathcal{M}$ that defines the set of strings matching the format. A *token table* containing plaintext-to-token mappings is stored in a database, and applications which need

*A preliminary version of this work was published at EUROCRYPT 2017. This is the full version.

access to data in the clear ask the database to do a lookup in this table for the plaintext corresponding to a particular token. Often applications reside in other organizations that have outsourced CCN management to a payments service. This technique can be viewed as a particularly inefficient FPE implementing a permutation $F_{K_\circ} : \mathcal{M} \to \mathcal{M}$ for a "key" $K_\circ$ that is a lazily constructed map of plaintexts to random ciphertexts (tokens).

Now that we have better approaches to FPE, a common problem faced by companies is upgrading from tokenization to an FPE scheme. This can be challenging when tokens have been distributed to various systems and users; there may be no way to recall the old tokens. The challenge here is therefore to build a new cipher that "completes" the domain of the cipher partially defined by the token table thus far.

A second example arises in the use of FPE for encryption of data before submission to restrictive cloud computing APIs. An instance of this arises with Salesforce, a cloud provider that performs customer relations management — at core they maintain on behalf of other companies databases of information about the companies' customers. As such, companies using Salesforce and desiring encryption of data before uploading have a large number of data fields with various format restrictions: email addresses, physical addresses, CCNs, names, phone numbers, etc. While we now have in-use solutions for defining formats via easy-to-use regular expressions [3, 11, 17], it is often the case that formats must change later. As a simple example: one may have thought only 16-digit CCNs were required, but later realized that 15-digit cards will need to be handled as well. Here we would like to, as easily as possible, modify an FPE $F_{K_\circ} : \mathcal{D} \to \mathcal{D}$ to one that works for an extended message space $\mathcal{M} \supset \mathcal{D}$. As with tokens and for similar reasons, it would be useful to maintain some old mappings under F in the new cipher.

Functionality-preserving modifications to encryption. The core challenge underneath both examples above is to take an existing cipher $F_{K_\circ}$ operating on some domain and, using knowledge of $K_\circ$, build a new cipher E_K such that $E_K(M) = F_{K_\circ}(M)$ for $M \in \mathcal{T} \subset \mathcal{M}$. We refer to $\mathcal{T}$ as the *preservation set*. In the tokenization example $\mathcal{T}$ could be the full set of messages for which entries in the table exist, and in the format-extension example $\mathcal{T}$ could be a subset of $\mathcal{D}$.

We note that trivial solutions don't seem to work. As already explained, the simplest solution of replacing old ciphertexts with new ones won't work when the old ciphertexts are unavailable (e.g., because other organizations have stored them locally). Furthermore, even when old ciphertexts can be revoked, the cost of decrypting and re-encrypting the whole database may be prohibitive.

Alternatively, consider encrypting a new point M in the following way. First check if $M \in \mathcal{T}$ and if so use the old cipher $F_{K_\circ}(M)$. Otherwise use a fresh key K for a new cipher E and apply $E_K(M)$. But this doesn't define a correct cipher, because different messages may encrypt to the same ciphertext: there will exist $M \notin \mathcal{T}$ and $M' \in \mathcal{T}$ for which $E_K(M) = F_{K_\circ}(M')$.

Our contributions. We explore for the first time functionality-preserving modifications to deployed ciphers. A summary of the settings and our constructions is given in Figure 1. Our main technical contribution is a scheme that we call the Zig-Zag construction. It can be used both in the tokenization setting and, with simple modifications, in the expanded format setting. It uses a new kind of cycle walking to define the new cipher on $\mathcal{M}$ using the old cipher $F_{K_\circ}$ and a helper cipher $\overline{E}_{\overline{K}} \colon \mathcal{M} \to \mathcal{M}$. The old mappings on points in $\mathcal{T}$ are preserved.

We analyze security of Zig-Zag in two cases, corresponding to the two situations discussed: (1) in domain completion, F has ciphertexts in the range $\mathcal{M}$ and (2) in domain extension, F works on a smaller domain $\mathcal{D} \subset \mathcal{M}$. For the first case, we show that the Zig-Zag construction is provably a strong pseudorandom permutation (SPRP) assuming that F and $\overline{E}$ both are themselves SPRPs. Extending to deal with tweaks [14] is straightforward.

The second case is more nuanced. We first observe that *no* scheme can achieve SPRP security

Setting	Description	Achievable security	Construction
Domain completion	Preserve partially defined cipher $\mathcal{T} \to \mathcal{M}$ in new cipher $\mathcal{M} \to \mathcal{M}$	SPRP	Zig-Zag
Domain extension	Extend cipher $\mathcal{D} \to \mathcal{D}$ to $\mathcal{M} \to \mathcal{M}$	SEPRP	Zig-Zag
	Extend cipher $\mathcal{D} \to \mathcal{D}$ to $\mathcal{M} \to \mathcal{M}$	SPRP (unknown $\mathcal{T}$)	Recursive Zig-Zag

Figure 1: Summary of different settings, security goals, and constructions. The set $\mathcal{M}$ is the new cipher's domain, the set $\mathcal{T}$ is the set of preserved points, and $\mathcal{D} \subset \mathcal{M}$ is the original domain in the extension setting.

when adversaries know $\mathcal{T}$. The attack is straightforward: query a point from $\mathcal{T}$ and see if the returned ciphertext is in $\mathcal{D}$ or not. Because it is functionality preserving, the construction must always have a ciphertext in $\mathcal{D}$, whereas a random permutation will do so only with probability $|\mathcal{D}|/|\mathcal{M}|$. Since we expect this ratio to usually be small, the attack will distinguish with high probability.

This begs the question of what security level is possible in this context. Investigating the attack ruling out SPRP security, we realize that the main issue is that ciphers that preserve points will necessarily leak to adversaries that the underlying plaintext is in $\mathcal{T}$. We formalize a slightly weaker security goal, called strong extended pseudorandom permutation (SEPRP) security in which a cipher must be indistinguishable from an ideal extended random permutation. Roughly this formalizes the idea that attackers should learn nothing but the fact that the distribution of points in $\mathcal{T}$ is slightly different from those in $\mathcal{M} \setminus \mathcal{T}$. We show that the Zig-Zag construction meets this new notion.

SEPRP security does leak more information to adversaries than does traditional SPRP security, and so we investigate the implications of this for applications. We formally relate SEPRP security to two security notions for FPE schemes from Bellare, Ristenpart, Rogaway, and Stegers [3], message recovery (MR) and message privacy (MP). We highlight the main results regarding MR here, and leave MP to the body. MR requires that an adversary, given the encryption of some challenge message as well as a chosen-plaintext encryption oracle, cannot recover the message with probability better than a simulator can, given no ciphertext and instead a test oracle that only returns one if the queried message equals the challenge. We show that there exist settings for which SEPRP security does *not* imply MR security, by way of an adversary whose success probability is 1, but any simulator succeeds with probability at most 1/2. The reason is that the adversary can exploit knowledge of membership in $\mathcal{T}$, whereas the simulator cannot.

This result may lead us to pessimistically conclude that SEPRP provides very weak security, but intuition states otherwise: an SEPRP-secure cipher should not leak more than one bit of information about a plaintext (whether or not it is in $\mathcal{T}$). The best MR attack one can come up with should therefore only have a factor of two speedup over attacking an SPRP-secure cipher. The gap between intuition and formalism lies in the strict way MR security was defined in [3]: simulators can only make as many queries as adversaries make and simulators receive nothing to aid in their attack. We therefore introduce a more general MR security notion that we uncreatively call generalized MR (gMR) security. The definition is now parameterized by both an auxiliary information function on the challenge plaintext as well as a query budget for simulators. We show that SEPRP security implies gMR security when the auxiliary information indicates whether the challenge is in $\mathcal{T}$ or not. We then show a general result that gMR security with this auxiliary information implies gMR security without any auxiliary information, as long as the simulator can make twice as many queries as the adversary. This makes precise the security gap between SEPRP and SPRP, and the interpretation is simply that adversaries get at most a factor of two speedup

in message recovery attacks.

Security and side-channel attacks. The Zig-Zag construction is not inherently constant time, which suggests it may be vulnerable to timing or other side-channel attacks. We prove in the body that timing leaks only whether a message is in $\mathcal{T}$ or not, and nothing further. We also discuss possible implementation approaches that avoid even this timing attack.

The Recursive Zig-Zag construction. Above we argued that in the domain extension setting SPRP security is unachievable should the adversary know (at least one) point in $\mathcal{T}$. In some scenarios, the attacker may be unable to learn which points are in $\mathcal{T}$, but is able to learn some information on $\mathcal{T}$ such as its size. This might arise, for example, should an attacker learn the size of a database but not have direct access to its contents. In this context the attack discussed above showing SPRP insecurity for all schemes no longer applies. We therefore explore feasibility of SPRP security in the domain extension setting when attackers know $|\mathcal{T}|$ but do not know $\mathcal{T}$.

First we show that SPRP security is still ruled out if the gap between the size of the old domain and the target domain is smaller than the number of new points by which the domain was extended, namely $|\mathcal{D}| - |\mathcal{T}| < |\mathcal{M} \setminus \mathcal{D}|$. To gain some intuition, consider the minimum number of points from $\mathcal{D}$ that map back to points in $\mathcal{D}$ for both an extended cipher and for a random permutation. For an extended cipher, at least $|\mathcal{T}|$ points are necessarily preserved, and so map to points in $\mathcal{D}$. For a random permutation, if the number of added points is large enough there is a probability that no point in $\mathcal{D}$ is mapped back to $\mathcal{D}$. Consequently, for a large enough $\mathcal{T}$ or when we add many points, the distribution of the number of points in $\mathcal{D}$ that map back to $\mathcal{D}$ differs sufficiently between extended ciphers and random permutations to give an adversary distinguishing advantage. For other ranges of parameters, however, with overwhelming probability a random permutation will have a subset of inputs that maps back to $\mathcal{D}$.

Unfortunately, the Zig-Zag construction does not meet SPRP security in this unknown $\mathcal{T}$ setting. Intuitively, the reason is that the construction biases the number of sets of size $|\mathcal{T}|$ that map to $\mathcal{D}$, with this bias growing as $|\mathcal{T}|$ grows.

We therefore provide a domain extension construction in the unknown $\mathcal{T}$ setting. It starts with a helper cipher $\overline{E}$ on $\mathcal{M}$, and utilizes the basic structure of the Zig-Zag to patch it in order preserve the points of $\mathcal{T}$. The patching occurs by replacing mappings for a t-size subset of $\mathcal{M}$ that $\overline{E}$ maps to $\mathcal{T}$. By patching those points in that set, as opposed to arbitrary points as in done in Zig-Zag, we preserve the distribution of sets of size $|\mathcal{T}|$ that are mapped to $\mathcal{D}$. To make this efficient we perform the patching recursively, handling the points in $\mathcal{T}$ one at a time, hence the name Recursive Zig-Zag for the construction. We prove that the construction works in expected time and space proportional (with a small constant) to $|\mathcal{T}|$, making it feasible for an application where $\mathcal{T}$ would need to be stored anyway, and analyze its SPRP security.

A ranking-based approach. An anonymous reviewer pointed out a potential alternative to our Zig-Zag construction that takes advantage of ranking functions, which are efficiently computable and invertible bijections from a domain $\mathcal{M}$ to $\mathbb{Z}_{|\mathcal{M}|}$. Ranking underlies many FPE constructions, and in some ways the reviewer's construction is simpler than Zig-Zag. The reviewer gave us permission to present the idea and discuss it in comparison to Zig-Zag. See Section 4.

Limitations and open problems. The approach we explore, of modifying a scheme after deployment, has several limitations. First, it requires the ability to perform membership tests against $\mathcal{T}$ and requires the old key K_{o} for the lifetime of the updated cipher. These must both be protected (in the former case, since it may leak some information about how people were using the cipher). In the case that $\mathcal{T}$ is an explicit list, one could cryptographically hash each point to provide some partial protection of plaintext data in case of key compromise, but this would only

provide marginal benefit in case of exposure since dictionary attacks would be possible.

Second, as points in $\mathcal{T}$ are submitted it would be convenient to gracefully remove points from it and "refresh" them with new mappings. This would be useful in the tokenization scenario should the client be able to update its token after a query. But there is no way to make "local" modifications to a cipher as any changed mapping necessarily affects at least one other domain point. We leave how to modify schemes gradually over time as an interesting open problem.

Our work only considered updating ciphers, but it could be that other cryptographic primitives might benefit from functionality-preserving updates. Future work could determine whether compelling scenarios exist, and what solutions could be brought to bear.

2 Preliminaries

Let $\mathcal{M}$ be a set, called the domain, and $\mathcal{K}$ be a set called the key space. Later we abuse notation and use sets to also denote efficient representations of them. A cipher is a family of permutations $E\colon \mathcal{K} \times \mathcal{M} \to \mathcal{M}$. This means that $E_K(\cdot) = E(K, \cdot)$ is a permutation for all $K \in \mathcal{K}$. Both E_K and its inverse E_K^{-1} must be efficient to compute. Block ciphers are a special case in which $\mathcal{M} = \{0,1\}^n$ for some integer n, and format-preserving encryption [3] is a generalization of ciphers that allows multiple lengths as well as tweaks [14]. Our results translate to that more general setting as well, but for the sake of simple exposition we focus on only a single domain, and use the term cipher instead of FPE. For a function f and set $\mathcal{X}$ that is a subset of its domain, we write $\mathsf{Img}_f(\mathcal{X})$ to denote the image of $\mathcal{X}$ under f, i.e., the set $\{f(x) | x \in \mathcal{X}\}$. We will give a concrete security treatment, meaning we will explicitly relay the running time (in some RAM model of computation) and number of queries made by adversaries. Each query is considered to be unit cost. We will use big-O notation $\mathcal{O}(\cdot)$ to hide only small constants that do not materially impact the interpretation of our results. We assume that adversaries do not repeat any oracle queries and do not ask queries to which they already know the answer, such as querying a decryption oracle with the result of a previous encryption oracle query.

SPRP security. Ciphers are considered secure if they behave like strong pseudorandom permutations (SPRPs). Let $\mathrm{Perm}(\mathcal{M})$ be the set of all permutations on any set $\mathcal{M}$. Consider a cipher $E\colon \mathcal{K} \times \mathcal{M} \to \mathcal{M}$. We define the *advantage* of an adversary $\mathcal{A}$ in distinguishing E and its inverse from a random permutation and its inverse as $\mathbf{Adv}_E^{\mathrm{sprp}}(\mathcal{A}) = \left|\Pr\left[\,\mathrm{SPRP1}_E^{\mathcal{A}} \Rightarrow 1\,\right] - \Pr\left[\,\mathrm{SPRP0}_E^{\mathcal{A}} \Rightarrow 1\,\right]\right|$. The two games SPRP1 and SPRP0 are defined in Figure 2 and the probabilities are taken over the random coins used in the games.

$\underline{\mathrm{SPRP1}_E^{\mathcal{A}}}$	$\underline{\mathrm{SPRP0}_E^{\mathcal{A}}}$
$K \xleftarrow{\$} \mathcal{K}$	$\pi \xleftarrow{\$} \mathrm{Perm}(\mathcal{D})$
$b' \leftarrow \mathcal{A}^{\mathrm{Enc},\mathrm{Dec}}$	$b' \leftarrow \mathcal{A}^{\mathrm{Enc},\mathrm{Dec}}$
return b'	return b'
$\underline{\mathrm{Enc}(M)}$	$\underline{\mathrm{Enc}(M)}$
return $E_K(M)$	return $\pi(M)$
$\underline{\mathrm{Dec}(C)}$	$\underline{\mathrm{Dec}(C)}$
return $E_K^{-1}(C)$	return $\pi^{-1}(C)$

Figure 2: SPRP security games for a cipher E.

The hypergeometric distribution. A hypergeometrically distributed random variable X has probability mass function

$$\Pr[\,X = k\,] = \frac{\binom{K}{k}\binom{N-K}{n-k}}{\binom{N}{n}}$$

where N is the total number of samples, K is the number of marked samples, n is the number of samples drawn, and k is the number of marked samples of the n total samples.

The hypergeometric distribution has very strong tail bounds. The following lemma, due to Hoeffding, gives a precise statement of this fact. A full proof of this lemma can be found in [9, 21] but is omitted here.

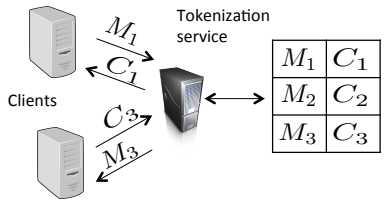

Figure 3: Tokenization system after choosing random values C_1, C_2, C_3 for plaintexts $\mathcal{T} = \{M_1, M_2, M_3\}$.

Lemma 1 *Let X be a hypergeometrically distributed random variable and n be the total number of samples drawn in the hypergeometric experiment described above. Then for any real number n, $\Pr[X \leq E[X] - vn] \leq e^{-2v^2 n}$, where e is the base of the natural logarithm.*

3 Extending Partially Used Message Spaces

We start by considering how to replace an existing cipher $F\colon \mathcal{K} \times \mathcal{M} \to \mathcal{M}$ with a new one $E\colon \mathcal{K} \times \mathcal{M} \to \mathcal{M}$, while maintaining backwards compatibility with the subset of the message space $\mathcal{T} \subset \mathcal{M}$ for which messages have already been encrypted. Our motivation for this originates with the following situation that arises in practice.

Updating tokenization deployments. Tokenization [23] is a set of techniques whose purpose is to provide confidentiality for relatively small data values (e.g., government ID numbers or credit card numbers). Usually tokenization is employed to meet regulatory requirements imposed by governments or industry standards bodies, such as the Payment Card Industry Security Standards Council (PCI) [10].

A tokenization system usually consists of a few parts: a server front-end which accepts tokenize/detokenize requests from authenticated clients, a cryptographic module that produces tokens for plaintext values, and a database backend that stores the plaintext/token mapping table. Each time a new tokenize request occurs for a plaintext M, a randomly generated value from $\mathcal{M}$ is chosen to be $F_{K_\mathsf{o}}(M)$. Here K_o is just an explicitly stored table of message, token pairs. The token $F_{K_\mathsf{o}}(M)$ is given back to the requester and stored for later use. Let $\mathcal{T} \subseteq \mathcal{M}$ be the set of all points for which tokens have been distributed. A diagram is shown in Figure 3.

Such tokenization systems are a bit clumsy. They do not scale very well, requiring protected storage linear in the number of plaintexts encrypted compared to FPE schemes that achieve this with just a small 128 bit key. (One cannot just store a key for a pseudorandom number generator and recreate values; this doesn't allow efficient decryption.) They also require precise synchronization to prevent two different tokens from being generated for the same input value. Since these systems often operate across entire continents, this is cumbersome. Companies therefore often want to move from tokenization to a modern solution using an FPE E.

One could choose a new key for E but the problem is then that the previously returned tokens will be invalidated, and this may cause problems down the road when clients make use of these tokens. Hence the desire to perform what we call *domain completion*: define $E_K(M)$ such that $E_K(M) = F_{K_\mathsf{o}}(M)$ for all $M \in \mathcal{T}$. This ensures that previously distributed tokens are still valid even under the new functionality. In deployment, any method for domain completion would most likely retain the token table, but the crucial difference in terms of performance is the immutability of the table. After switching to the keyed cipher, the table can be made read-only and distributed with the FPE software as a file with no expensive and complicated database needed to ensure consistency and availability. In most contexts, one will want to keep the file secret since it may

leak information about previous use of F.

Domain completion, formally. A domain completion setting is defined to be a tuple $(F, \mathcal{M}, \mathcal{T})$ consisting of a cipher F with domain $\mathcal{M}$ and the preservation set $\mathcal{T} \subseteq \mathcal{M}$. Relative to some fixed domain completion setting (that later will always be made clear from context), a domain-completed cipher $\mathrm{DCC} = (KT, E)$ consists of an algorithm and a cipher. The algorithm is called a domain-completion key transform. It is randomized, takes as input a key K_{o} for F and the preservation set $\mathcal{T}$, and outputs a key for the cipher E. The cipher is assumed to have a key space compatible with the output of KT. For some preservation set $\mathcal{T}$, the induced key generation algorithm for E consists of choosing a random key K_{o} for F, running $KT(K_{\mathsf{o}}, \mathcal{T})$ and returning the result.

A domain-completed cipher DCC preserves a point M if $E_K(M) = F_{K_{\mathsf{o}}}(M)$ with probability one over the experiment defined by running the induced key generation for E. We say that KT preserves a set $\mathcal{T}$ if it preserves each $M \in \mathcal{T}$. The ability of a key transformation to achieve preservation implies that K must somehow include (an encoding of) K_{o}. In the case where F is a tokenization system, then K_{o} is a table of at least $t = |\mathcal{T}|$ points.

We measure security for a domain-completed cipher via the SPRP advantage of the cipher E using its induced key generation algorithm. We will quantify over all preservation sets, that is, over all sets of size t.

4 Domain Completion via Rank-Encipher-Unrank

An anonymous reviewer pointed out an approach to domain completion for schemes constructed using the rank-encipher-unrank approach of [3]. With permission we reproduce it here. Recall that a rank-encipher-unrank construction uses a ranking function $\mathit{rank} : \mathcal{M} \to \mathbb{Z}_m$, which is a bijection with inverse $\mathit{unrank} : \mathbb{Z}_m \to \mathcal{M}$. Both must be efficiently computable. One additionally uses cipher $\overline{E}$ that operates on domain $\mathbb{Z}_m$. (This is referred to as an integer FPE in [3].) Then one enciphers a point $X \in \mathcal{M}$ via $\mathit{unrank}(\overline{E}_K(\mathit{rank}(X)))$ and decrypts a point Y via $\mathit{unrank}(\overline{D}_K(\mathit{rank}(Y)))$.

Now consider a domain completion setting $(F, \mathcal{M}, \mathcal{T})$. Let $\mathcal{D} = \mathcal{M} \setminus \mathcal{T}$ be the set of domain points not in the preservation set. Let $\mathcal{R} = \mathcal{M} \setminus \mathsf{Img}_{F_{K_{\mathsf{o}}}}(\mathcal{T})$, where $\mathsf{Img}_{F_{K_{\mathsf{o}}}}(\mathcal{T}) = \{F_{K_{\mathsf{o}}}(X) \mid X \in \mathcal{T}\}$, be the set of range points not in the image of $F_{K_{\mathsf{o}}}$ on $\mathcal{T}$. The construction uses a cipher $\overline{E} : \mathbb{Z}_d \to \mathbb{Z}_d$ and a ranking function $\mathit{rank} : \mathcal{M} \to \mathbb{Z}_m$ with inverse unrank. The construction builds from rank rankings $\mathit{rank}_{\mathcal{D}} : \mathcal{D} \to \mathbb{Z}_d$ and $\mathit{rank}_{\mathcal{R}} : \mathcal{R} \to \mathbb{Z}_d$. It then encrypts by checking if a point X is in $\mathcal{T}$ and, if so, outputting $F_{K_{\mathsf{o}}}(X)$ and otherwise outputting $\mathit{unrank}_{\mathcal{R}}(\overline{E}_{\overline{K}}(\mathit{rank}_{\mathcal{D}}(X)))$.

In detail, the domain-completed cipher $\mathrm{RTE} = (KT^{rte}, E^{rte})$ is defined as follows. The domain-completion key transform $KT^{rte}(K_{\mathsf{o}}, \mathcal{T})$ first computes the set $\mathsf{Img}_{F_{K_{\mathsf{o}}}}(\mathcal{T})$. Then it computes the set $\{\mathit{rank}(X) \mid X \in \mathcal{T}\}$ and sorts it to obtain a list $\vec{x} = (\vec{x}_1, \ldots, \vec{x}_t) \in \mathbb{Z}_m^t$. Similarly it computes $\{\mathit{rank}(Y) \mid Y \in \mathsf{Img}_{F_{K_{\mathsf{o}}}}(\mathcal{T})\}$ and sorts it to obtain a list $\vec{y} = (\vec{y}_1, \ldots, \vec{y}_t) \in \mathbb{Z}_m^t$. It also chooses a new key $\overline{K}$ for a helper cipher $\overline{E}$ on $\mathbb{Z}_d$ and outputs $K = (K_{\mathsf{o}}, \overline{K}, \vec{x}, \vec{y})$.

Enciphering is performed via $E^{rte}_K(X) = \mathit{unrank}_{\mathcal{R}}(\overline{E}_{\overline{K}}(\mathit{rank}_{\mathcal{D}}(X)))$ for rankings defined as follows. The first ranking, $\mathit{rank}_{\mathcal{D}}(X)$, works for $X \in \mathcal{D}$ by computing $x \leftarrow \mathit{rank}(X)$, then determining, via a binary search, the largest index i such that $\vec{x}_i < x$, and finally outputting $x - i$. The inverse of $\mathit{rank}_{\mathcal{D}}$ is $\mathit{unrank}_{\mathcal{D}}(x')$. It works for $x' \in \mathbb{Z}_d$ by using a binary search to determine the largest index j such that $\vec{x}_j - j + 1 \le x'$, and then outputting $X \leftarrow \mathit{unrank}(x' + j)$. The construction of $\mathit{rank}_{\mathcal{R}}$ is similar, using $\vec{y}$ instead of $\vec{x}$.

This domain-completed cipher can be shown to be SPRP secure and, looking ahead, one can simply adapt it to the domain extension case to achieve SEPRP security (as defined in Section 6). This approach relies on having a ranking for $\mathcal{M}$. While not all languages have efficient rankings [3],

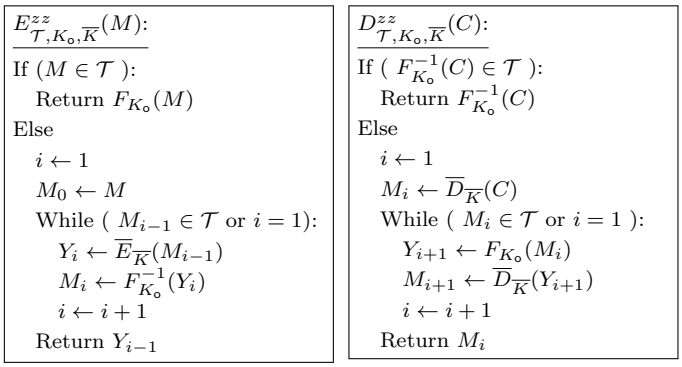

Figure 4: Zig-Zag encryption and decryption algorithms.

efficient rankings can be built for most formats of practical interest [3, 11, 17, 18]. The additional overhead of removing the $\mathcal{T}$ (resp. $\mathsf{Img}_{F_{K_o}}(\mathcal{T})$) points requires time proportional to $\log t$ and space equal to $2t$ multiplied by some representation-specific constant.

Our construction, to be presented in the next section, avoids the extra space requirements and the binary search. It only requires the ability to determine membership in $\mathcal{T}$, which enables more flexibility in implementation choices, such as using an API to check membership or representing $\mathcal{T}$ via a compact Bloom filter. It also allows, via precomputation, a constant-time implementation using table look-up (assuming constant time implementations of F and $\overline{E}$).

We note that the straightforward implementation of both approaches leaks, via timing side-channels, whether a domain point is in $\mathcal{T}$. The ranking-based approach may leak more with a naive implementation of binary search. Ranking itself may in some cases be tricky to implement without side-channels, if one uses the table-based constructions for ranking regular languages [3, 11, 17] or context-free grammars [18].

5 The Zig-Zag Construction for Domain Completion

In this section we will introduce an algorithm that achieves SPRP security for domain completion. Fix a domain completion setting $(F, \mathcal{M}, \mathcal{T})$. Then the Zig-Zag domain-completed cipher $\mathrm{ZZ} = (KT^{zz}, E^{zz})$ is defined as follows. The key transform KT^{zz} takes inputs K_o and $\mathcal{T}$ and outputs the tuple $(\mathcal{T}, K_o, \overline{K})$ where $\overline{K}$ is a randomly chosen key for a cipher $\overline{E}$ on domain $\mathcal{M}$. We refer to $\overline{E}$ as the helper cipher. The triple $(\mathcal{T}, K_o, \overline{K})$ define a key for the enciphering algorithm E^{zz} and deciphering algorithm D^{zz}, detailed in Figure 4.

The Zig-Zag construction is simple. For points in $\mathcal{T}$ it behaves like F_{K_o}. On all other points it behaves like $\overline{E}_{\overline{K}}$, save those points whose images under $\overline{E}_{\overline{K}}$ are in $F_{K_o}(\mathcal{T})$. Those points need images under our permutation, so we use the (unused) points in $\overline{E}_{\overline{K}}(\mathcal{T})$. The inner while loop does this re-assignment.

Towards building intuition about why the Zig-Zag construction works, we start by discussing why traditional cycle walking will not work for our context. Cycle walking is a generic method for achieving format-preserving encryption on a set by re-encrypting an input point until it falls in a desired subset of the domain of the cipher [7]. Cycle-walking could ostensibly be used instead of zig-zagging as follows. Consider an input point $X \in \mathcal{M} \backslash \mathcal{T}$, and suppose that $Y = \overline{E}_{\overline{K}}(X) \in \mathsf{Img}_{F_{K_o}}(\mathcal{T})$. Then since Y is already a point required for the preservation set, we can't map it to X, and instead cycle walk by computing $Y' = \overline{E}_{\overline{K}}(Y)$, $Y'' = \overline{E}_{\overline{K}}(Y')$ and so on, stopping the first time we find a point not in the image of $\mathcal{T}$ and having X map to that final value. But the problem is that, unlike

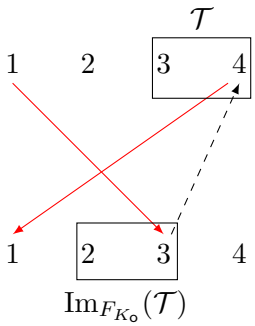

Figure 5: Graphical depiction of a zig-zag. The domain (first row) has a target set $\mathcal{T} = \{3, 4\}$. We encrypt 1 to 3, which collides with the ciphertext in the image of $\mathcal{T}$. We then decrypt to get $4 \in \mathcal{T}$ and re-encrypt 4 to get 1.

with traditional cycle walking, the intermediate points Y, Y', Y'' are themselves valid *inputs* to the cipher, and using them for the cycle walk obviates using the obvious mapping of, e.g., $\overline{E}_{\overline{K}}(Y)$.

The Zig-Zag avoids this problem by only trying a different point of $\mathcal{T}$ each time $\overline{E}$ returns a point in $\mathsf{Img}_F(\mathcal{T})$. This ensures that as we do our search for a point to which we will map the input X, we are only using $\overline{E}$ on points $Y, Y', Y'' \in \mathcal{T}$. A diagram depicting this process appears in Figure 5. There $\mathcal{M} = \{1, 2, 3, 4\}$ and $\mathcal{T} = \{3, 4\}$. The solid red lines signify encryption by $\overline{E}_{\overline{K}}$ and the dashed black line represents $F_{K_o}^{-1}$. We start by calling $\overline{E}_{\overline{K}}(1)$, which (say) gives us the image of a point in $\mathcal{T}$. We call $F_{K_o}^{-1}$ and find that the preimage of this point is 4. We then call $\overline{E}_{\overline{K}}(4)$, which gives us $E_{\overline{K}}^{zz}(1) = \overline{E}_{\overline{K}}(4) = 1$.

5.1 Running time of the Zig-Zag construction

Next we'll prove a few results about the running time of the Zig-Zag construction. In this section we will only analyze encryption and omit decryption for the sake of simplicity. All results in this section hold for decryption as well. The first result, stated in Lemma 2, will upper bound the worst-case runtime of the Zig-Zag: for a sequence of q encryptions, at most $q + |\mathcal{T}|$ iterations of the inner while loop will occur. We formalize this using a simple pseudocode game ENC-Runtime, shown in Figure 6. The game ENC-Runtime is parameterized by the target domain $\mathcal{T}$, a pair of permutations π_1 and π_2, and a vector of messages $\vec{q}$. The game simply carries out the Zig-Zag algorithm on each message in $\vec{q}$ and records the total number of loop iterations to encrypt all messages in $\vec{q}$. Note that Lemma 2 also proves the Zig-Zag cannot loop infinitely. Because it is a correctness result we eschew specifying the distribution over permutations and simply use arbitrary permutations π_1 and π_2.

Lemma 2 *Let $\mathcal{T} \subset \mathcal{M}$. Let $\vec{q}$ be any sequence of q distinct elements of $\mathcal{M} \setminus \mathcal{T}$. Let π_1 and π_2 be any permutations on $\mathcal{M}$. Let Z be the output of* ENC-Runtime$(\mathcal{T}, \pi_1, \pi_2, \vec{q})$ *in Figure 6. Then $Z \leq q + |\mathcal{T}|$.*

Proof: Let $\mathbb{M}_i = \{M_{i,1}, M_{i,2}, \ldots, M_{i,\ell_i}\}$ be the sequence of points output by π_1^{-1} while encrypting point $M_i \in \vec{q}$. Below we will sometimes refer to M_i as $M_{i,0}$, the "zeroth" element of the sequence. The number of points in this sequence, ℓ_i, is also equal to the number of loop iterations that occurred while encrypting M_i. The total number of loop iterations taken for all q points is simply the sum, $Z = \sum_{i=1}^{q} \ell_i$. Note that $\ell_i \geq 1$ for all i, because the loop always executes at least once.

We are abusing notation slightly by representing these sequences as finite, but momentarily we will prove the sequences are always finite.

To prove the theorem we need to prove two things about the $\mathbb{M}_i$ sequences: (1) that no $\mathbb{M}_i$ contains any point of $\mathcal{T}$ more than once, and (2) that each point of $\mathcal{T}$ occurs in at most one $\mathbb{M}_i$. Start by proving (1). To see this, assume for contradiction $M_{i,j} = M_{i,k}$ for $j < k$ for some $\mathbb{M}_i$. By the permutivity of π_1^{-1} and π_2, this implies $M_{i,j-1} = M_{i,k-1}$. Repeating this argument j times gives that $M_{i,0} = M_{i,k-j+1}$ where $M_{i,0}$ is the original message $M_i \in \vec{q}$. But this is a contradiction: every element of $\vec{q}$ is in $\mathcal{M} \setminus \mathcal{T}$ and $M_{i,k-j+1}$ is in $\mathcal{T}$.

To prove (2), assume for contradiction that the same point of $\mathcal{T}$ appears at position j in sequence $\mathbb{M}_i$ and position l in sequence $\mathbb{M}_k$ (i.e., $M_{i,j} = M_{k,l}$). If $M_{i,j-1} \neq M_{k,l-1}$ we are done because this violates the permutivity of π_1^{-1} and π_2. If $M_{i,j-1} = M_{k,l-1}$, repeat until they are not equal. This must occur at some point: if $j < l$ we will eventually get $M_{i,0} = M_{k,l-j+1}$, which is a contradiction because $M_{i,0} \in \mathcal{M} \setminus \mathcal{T}$ but $M_{k,l-j+1}$ is in $\mathcal{T}$. The same argument applies if $j > l$: we eventually get $M_{i,j-l+1} = M_{k,0}$. If $j = l$ we eventually get $M_{i,0} = M_{k,0}$, which is a contradiction because the elements of $\vec{q}$ are distinct.

With (1) and (2), we know that each element of $\mathcal{T}$ occurs at most once in at most one sequence $\mathbb{M}_i$. Each $\mathbb{M}_i$ contains exactly one point of $\mathcal{M} \setminus \mathcal{T}$ (the last point M_{i,ℓ_i}) so there are exactly q points of $\mathcal{M} \setminus \mathcal{T}$ and at most $|\mathcal{T}|$ points of $\mathcal{T}$ across all the sequences, so $Z = \sum_{i=1}^{q} \ell_i \leq q + |\mathcal{T}|$ as desired. ∎

```
ENC-Runtime(T, π1, π2, q⃗):
Z ← 0
For M in q⃗:
  If (M ∈ T ):
    Continue
  Else
    i ← 1
    M0 ← M
    While ( Mi−1 ∈ T or i = 1):
      Yi ← π2(Mi−1)
      Mi ← π1^{-1}(Yi)
      i ← i + 1
    Z ← Z + i
Return Z
```

Figure 6: Game for measuring runtime of Zig-Zag encryption.

In the following lemma we also show that, when encrypting the entire domain $\mathcal{M}$, $\overline{E}$ is executed at most once per point in $\mathcal{M}$. This means that (roughly speaking) the aggregate cost of enciphering the entire domain under Zig-Zag is almost the same as enciphering with $\overline{E}$, assuming $|\mathcal{T}| \ll |\mathcal{M}|$. Otherwise, it's at most twice the cost. Here, as above, we replace $\overline{E}$ with π_2 and F with π_1, arbitrary permutations over $\mathcal{M}$.

Lemma 3 *Let $\mathcal{T} \subset \mathcal{M}$ and $\vec{q}$ be a vector in which each point of $\mathcal{M} \setminus \mathcal{T}$ appears exactly once. Let $\mathbb{M}$ be the sequences of points input to π_2 during the execution of* ENC-Runtime$(\mathcal{T}, \pi_1, \pi_2, \vec{q})$ *in Figure 6, where $\mathbb{M}_i$ is the sequence for the ith element of $\vec{q}$. Then (1) $\mathbb{M}$ contains no duplicates, and (2) $|\mathbb{M}| \leq |\mathcal{M}|$.*

Proof: Before we begin, note that $|\mathbb{M}|$ would not change if $\vec{q}$ included all of $\mathcal{M}$ instead of only $\mathcal{M} \setminus \mathcal{T}$—the permutation π_2 is never called if the point being encrypted is in $\mathcal{T}$.

We begin by proving (1). First we examine the structure of these sequences. The first point in each $\mathbb{M}_i$ is distinct and $|\mathbb{M}_i| \geq 1$ for all i. By Lemma 2, no individual list contains duplicate points of $\mathcal{T}$, and a similar argument shows that no individual list contains duplicate points of $\mathcal{M} \setminus \mathcal{T}$. Again by a similar argument to the one used in Lemma 2, we can see that no point of $\mathcal{M} \setminus \mathcal{T}$ appears in more than one list; otherwise permutivity would be violated. To prove (2), apply Lemma 2 with $q = |\mathcal{M}| - |\mathcal{T}|$. ∎

The previous two results are worst-case, assuming no distribution over the permutations π_1 and π_2. They imply that in the worst case, the runtime of the Zig-Zag is quite bad—a single encryption or decryption could require as many as $2|\mathcal{T}|$ calls to π_1 and π_2! The next two results will show

REAL-Runtime$(\vec{q}, \mathcal{T})$:	IDEAL-Runtime$(\vec{q}, \mathcal{T})$:
$K_{\mathrm{o}} \leftarrow\!\!\$\ \mathcal{K}$	$\Pi_1 \leftarrow\!\!\$\ \mathrm{Perm}(\mathcal{M})$
$\overline{K} \leftarrow\!\!\$\ \mathcal{K}$	$\Pi_2 \leftarrow\!\!\$\ \mathrm{Perm}(\mathcal{M})$
$Z \leftarrow$ ENC-Runtime$(\mathcal{T}, F_{K_{\mathrm{o}}}, \overline{E}_{\overline{K}}, \vec{q})$	$Z \leftarrow$ ENC-Runtime$(\mathcal{T}, \Pi_1, \Pi_2, \vec{q})$
Return Z	Return Z

Figure 7: Runtime games for Theorem 1 and Theorem 2.

that in practice, where the Zig-Zag would be instantiated with SPRPs, its average performance is very good and the worst-case behavior only occurs with exponentially small probability. The first result shows the expected running time for an arbitrary point is small, as captured by the next theorem, and assuming the underlying ciphers are sampled uniformly at random from the space of all permutations over $\mathcal{M}$.

Theorem 1 *Let $\vec{q}$ be a one-element vector containing an arbitrary point in $\mathcal{M} \setminus \mathcal{T}$. Let $\mathcal{T} \subset \mathcal{M}$, and let $|\mathcal{T}| = t$. Let Z be a random variable (over the random choice of Π_1 and Π_2) denoting the output of game* IDEAL-Runtime$(\vec{q}, \mathcal{T})$ *in Figure 7. Then, if $t \leq |\mathcal{M}|/2$, it holds that $\mathrm{E}[\,Z\,] \leq 2$.*

Proof: Since we assume the point is not in $\mathcal{T}$, clearly we have that $\Pr[Z > 1] = 1$. By correctness, we also know that $\Pr[Z > t] = 0$. Then for any $1 \leq j \leq t$, because Π_1, Π_2 are random permutations independent of M, we have that

$$\Pr[\,Z = j\,] = \left[\prod_{i=1}^{j-1} \frac{(t-i+1)}{(m-i+1)}\right] \cdot \left(1 - \frac{t-j+1}{m-j+1}\right) .$$

Letting $P_j = \prod_{i=1}^{j-1} \frac{t-i+1}{m-i+1}$, we can plug into the definition of expectation to get that

$$\mathrm{E}[\,Z\,] = \sum_{j=1}^{t} j \left[P_j - P_j \cdot \frac{t-j+1}{m-j+1}\right] = \sum_{j=1}^{t} jP_j - \sum_{j=1}^{t} jP_{j+1}$$

where we've used the fact $P_j \cdot \frac{t-j+1}{m-j+1} = P_{j+1}$. Investigating the right-hand side of the equation, we have that the left summand is one factor of P_j larger than the right summand when the index of summation on the left is one greater than on the right. Thus, for every P_j there will be a jP_j term in the overall summation and a $-(j-1)P_j$ term, so every term of the right summation is cancelled by a term of the left summation except for the final one, tP_{t+1}. Thus we can rewrite the equation to get

$$\mathrm{E}[\,Z\,] = \left[\sum_{j=1}^{t} P_j\right] - tP_{t+1} \leq 1 + \sum_{j=2}^{t} \frac{1}{2^{j-1}} .$$

To justify the final inequality, observe that the first term of the summation is the empty product, which is by definition equal to 1. For the second summation, noticing that for $2 \leq j \leq t$, plugging $t = \frac{m}{2}$ into P_j gives us a summand which is upper-bounded by $\frac{1}{2^{j-1}}$. The rightmost term is bounded above by 1 so $\mathrm{E}[\,Z\,] \leq 2$ for $t \leq \frac{m}{2}$. ∎

For simplicity in the above we restricted to $t \leq m/2$. For larger t, i.e. $\frac{m}{2} \leq t \leq m-1$, a similar analysis can be done using the sum of a geometric series with ratio $\frac{1}{2} \leq \frac{t}{m} < 1$ in the final step instead of $\frac{1}{2}$. The same bound can be obtained for the run time of D^{zz}.

Running time of the Zig-Zag with SPRPs. The previous bound on the expectation is information-theoretic — it only holds if F and $\overline{E}$ are random permutations. To bound the running

time of the Zig-Zag when instantiated with SPRPs, we need to do a reduction. To obtain sharp bounds, rather than measuring a single point we will measure the cumulative running time of enciphering an arbitrary sequence of points in $\mathcal{M} \setminus \mathcal{T}$.

Theorem 2 *Let Z_{real} be a random variable (over the random choices of K_{o} and $\overline{K}$) denoting the output of game* REAL-Runtime *in Figure 7. Let $\vec{q}$ be a sequence of q arbitrary points in $\mathcal{M} \setminus \mathcal{T}$. Let $\mathcal{T} \subset \mathcal{M}$. Let $t = |\mathcal{T}|$ and assume $t \leq |\mathcal{M}|/2$. Then we give adversaries $\mathcal{B}$ and $\mathcal{C}$ so that*

$$\Pr[\, Z_{\text{real}} > 8q \,] \leq \mathbf{Adv}^{\text{sprp}}_{F}(\mathcal{B}) + \mathbf{Adv}^{\text{sprp}}_{\overline{E}}(\mathcal{C}) + e^{-\frac{9}{4}q} \,.$$

The adversaries $\mathcal{B}$ and $\mathcal{C}$ both make at most $8q$ queries to their oracles and run in $\mathcal{O}(q)$ time.

Intuitively, this result shows that the Zig-Zag will be nearly as efficient in the "real" world as it would be if it was instantiated with random permutations. In particular, it shows that if $\overline{E}$ and F are both SPRPs the amortized cost of a Zig-Zag encryption is constant except with very small probability. For ease of exposition this bound is slightly loose; a slightly more complicated proof could reduce the constant 8 to obtain a smaller v in the last step.

Proof: Begin by defining another random variable, Z_{ideal}, in the probability space defined by the random choices of Π_1 and Π_2 in game IDEAL-Runtime in Figure 7, denoting the output of game IDEAL-Runtime. The difference in the probability of games REAL-Runtime and IDEAL-Runtime outputting more than $8q$ is $\Pr[\, Z_{\text{real}} > 8q \,] - \Pr[\, Z_{\text{ideal}} > 8q \,]$. Now, construct an intermediate game by replacing F in REAL-Runtime by Π_1, a random permutation on $\mathcal{M}$. Call the output of this intermediate game Z_{mid}. Clearly,

$$\begin{aligned} \Pr[\, Z_{\text{real}} > 8q \,] - \Pr[\, Z_{\text{ideal}} > 8q \,] = {} & \Pr[\, Z_{\text{real}} > 8q \,] - \Pr[\, Z_{\text{mid}} > 8q \,] \\ & + \Pr[\, Z_{\text{mid}} > 8q \,] - \Pr[\, Z_{\text{ideal}} > 8q \,] \,. \end{aligned} \tag{1}$$

To prove the theorem, it suffices to upper-bound the two terms on the right-hand side as well as $\Pr[\, Z_{\text{ideal}} > 8q \,]$.

We will start with upper-bounding $\Pr[\, Z_{\text{real}} > 8q \,] - \Pr[\, Z_{\text{mid}} > 8q \,]$, which we will do by constructing an SPRP adversary $\mathcal{B}$ against F. The adversary $\mathcal{B}$ simulates the game REAL-Runtime using its oracles instead of F. If at any point $Z > 8q$ during its simulation it outputs 1, else 0. By construction,

$$\Pr[\, Z_{\text{real}} > 8q \,] - \Pr[\, Z_{\text{mid}} > 8q \,] \leq \mathbf{Adv}^{\text{sprp}}_{F}(\mathcal{B}) \,.$$

Next, we will upper-bound the quantity $\Pr[\, Z_{\text{mid}} > 8q \,] - \Pr[\, Z_{\text{ideal}} > 8q \,]$ using an SPRP adversary $\mathcal{C}$ against $\overline{E}$. The adversary $\mathcal{C}$ simulates Π_1 by sampling a random point of $\mathcal{M}$ for each point in $\mathcal{T}$ subject to permutivity. Then it simulates the intermediate game described above using its encryption oracle in place of $\overline{E}$ and with its queries as the vector $\vec{q}$. If at any point $Z > 8q$ it outputs 1, else 0. As above, it is true by construction that

$$\Pr[\, Z_{\text{mid}} > 8q \,] - \Pr[\, Z_{\text{ideal}} > 8q \,] \leq \mathbf{Adv}^{\text{sprp}}_{\overline{E}}(\mathcal{C}) \,.$$

To complete the proof we only need to bound $\Pr[\, Z_{\text{ideal}} > 8q \,]$. The inner while loop in game IDEAL-Runtime will terminate when Π_2 outputs a point in $\mathcal{M} \setminus \text{Im}_{\Pi_1}(\mathcal{T})$. For all q loops to terminate we need at least q points in this set. We can model this probability space as an urn experiment where points in $\mathcal{M} \setminus \text{Im}_{\Pi_1}(\mathcal{T})$ are "marked" and points in $\text{Im}_{\Pi_1}(\mathcal{T})$ are "unmarked". Each loop iteration samples a point without replacement, so we can use the tail bound on the hypergeometric distribution from Lemma 1 to upper bound the probability of getting fewer than q marked points in $8q$ samples. Let J be a random variable denoting the number of marked

points returned by Π_2 after $8q$ queries. Then, $\Pr[\, Z_{\text{ideal}} > 8q \,] = \Pr[\, J < q \,]$. Since $t \leq |\mathcal{M}|/2$, $\mathrm{E}[\, J \,] = 8q\frac{t}{|\mathcal{M}|} \leq 4q$. Plugging this into the bound, we get that for any v

$$\Pr[\, J < 4q - 8vq \,] \leq e^{-2v^2(8q)} \; .$$

Setting $v = \frac{3}{8}$ gives us $J < q$ on the left-hand side, so plugging this in and doing some arithmetic gives us

$$\Pr[\, J < q \,] \leq e^{-\frac{9}{4}q} \;\Rightarrow \Pr[\, Z_{\text{ideal}} > 8q \,] \leq e^{-\frac{9}{4}q} \; .$$

Combining and rewriting Equation 1 above gives us the desired bound on $\Pr[\, Z_{\text{real}} > 8q \,]$. ∎

5.2 Security of the Zig-Zag construction

In the domain completion setting, our Zig-Zag construction achieves SPRP security. We prove the following theorem in Appendix A. Our proof proceeds via a move to an information-theoretic setting in which F and $\overline{E}$ are replaced by random permutations. For simplicity in these reductions, we will use the worst-case upper bound on the number of queries required (i.e., $q + |\mathcal{T}|$). A tighter bound on the number of queries can be obtained with an analysis similar the one in Theorem 2, at the cost of some small decrease in the reduction's success probability. Finally, we perform an analysis to show that the Zig-Zag domain-completed cipher, when using random permutations, exactly defines a random permutation.

Theorem 3 *Fix a domain completion setting $(F, \mathcal{M}, \mathcal{T})$ and let $\mathrm{ZZ} = (KT^{zz}, E^{zz})$ be the Zig-Zag domain-completed cipher using helper cipher $\overline{E}$. Let $\mathcal{A}$ be an SPRP adversary making at most q queries to its oracles. Then the proof gives explicit adversaries $\mathcal{B}$ and $\mathcal{C}$ such that*

$$\mathbf{Adv}^{\text{sprp}}_{E^{zz}}(\mathcal{A}) \leq \mathbf{Adv}^{\text{sprp}}_{\overline{E}}(\mathcal{B}) + \mathbf{Adv}^{\text{sprp}}_{F}(\mathcal{C}) \; .$$

Adversaries $\mathcal{B}$ and $\mathcal{C}$ each run in time that of $\mathcal{A}$ plus at most a $\mathcal{O}(q) + |\mathcal{T}|$ overhead and each make at most $q + |\mathcal{T}|$ queries.

6 Domain Extenders for Deployed Ciphers

We now look at a distinct but related setting in which we want to extend the message space of a cipher after it has been deployed. Suppose we have an FPE $F_{K_\circ}$ for some message space $\mathcal{D}$ that has already been used to encrypt a number of plaintexts. We later learn that we need to be able to encrypt as well plaintexts from a larger format $\mathcal{M} = \mathcal{N} \cup \mathcal{D}$.

Practical motivations for domain extension. While perhaps odd at first, message space extension arises frequently in deployment. An example is the use of encryption schemes in settings with constrained formatting on ciphertexts, such as the traditional credit card number example. Say we have deployed an FPE scheme for 16-digit credit card numbers only to later realize we must handle 15-digit credit card numbers as well. In this case it might be that $|\mathcal{D}| = 10^{15}$, $|\mathcal{N}| = 10^{14}$ and $|\mathcal{M}| = 10^{15} + 10^{14}$. (Recall that the last digit of a credit card number is a checksum, so a 15 digit CCN is only 14 digits of information.)

In deployment, such a format change is often precipitated by one of two things: a change in customer requirements or a change in application behavior. Changes in customer requirements often occur when businesses adopt FPE incrementally, rather than all at once. For example, a business might deploy FPE for users in the United States first, then later deploy it for users in China as well. If the format of the FPE was English-only initially, the inclusion of Chinese users will necessitate a change to this format. Sometimes customer requirements change because of external

changes in their industries. When computerized gift cards gained widespread adoption, the credit card industry had to modify its standard for assigning credit card numbers to include a reserved range for numbers corresponding to temporary gift cards.

Changes in application behavior are problematic for businesses that use FPE in conjunction with cloud-hosted software. When FPE is deployed in such a setting (in which, it is important to note, the users have no control over application behavior) the formats are chosen to hew as closely as possible to the format validation used by the application. If the software vendor changes the way formats are validated, the FPE format must change as well or leave businesses with an unusable application.

We can achieve the desired security trivially by using an FPE on $\mathcal{M}$ with a fresh key. But this requires retrieving, decrypting, and re-encrypting all ciphertexts already stored under the old format and key. In many contexts this is rather expensive, and may not even be feasible should the ciphertexts be unavailable for update (e.g., because they've been handed back to some client's systems as a token and no API exists for recalling them).

One way to handle this extension would be to use a separate FPE for 16-digit credit card numbers and for 15-digit credit card numbers. The security of this kind of solution is, a priori, unclear, as such a ciphertext later accessible to adversaries will trivially leak which portion of the message space a plaintext sits. We will analyze the security of this formally below.

We can do better in the case that we have only used the old FPE on a small portion $\mathcal{T} \subset \mathcal{D}$ of the old message space. We'd like to preserve the decryptability of the points in $\mathcal{T}$ while otherwise picking mappings that are indistinguishable from a random permutation. We will formalize this goal below.

It may seem odd to assume that a list of already-encrypted points $\mathcal{T}$ is obtainable. After all, if we can extract a list of previously encrypted values, why can't we simply download and re-encrypt them? As discussed above, it is often difficult to authoritatively (i.e. ensuring all parties agree) change any value in a complex software system after it has been created. It's also possible a description of $\mathcal{T}$ (like a regular expression) may be stored in a concise form in some application metadata that is stored on the encryption server.

Domain extension, formally. A domain extension setting is defined as a tuple $(F, \mathcal{D}, \mathcal{T}, \mathcal{M})$ consisting of a cipher F on domain $\mathcal{D}$, an extended domain $\mathcal{M}$, and a preservation set $\mathcal{T} \subseteq \mathcal{D}$. A domain extended cipher $\mathrm{DEC} = (KT, E)$ is an algorithm and a cipher. The randomized algorithm KT, called a domain extension key transformation, takes as input a key K_{o} for F and the preservation set $\mathcal{T}$, and outputs a key K for the cipher E whose domain is $\mathcal{M}$. The cipher E is assumed to have a key space compatible with the output of KT. For some preservation set $\mathcal{T}$, the induced key generation algorithm for E consists of choosing a random key K_{o} for F, running $KT(K_{\mathsf{o}}, \mathcal{T})$ and returning the result.

A domain-extended cipher DEC preserves a point M if $E_K(M) = F_{K_{\mathsf{o}}}(M)$ with probability one over the experiment defined by running the induced key generation for E. We say that DEC preserves a set $\mathcal{T}$ if it preserves each $M \in \mathcal{T}$.

Impossibility of SPRP security. The most obvious security goal for a domain-completed cipher $\mathrm{DEC} = (KT, E)$ is for all efficient adversaries to have low SPRP advantage against the cipher E using its induced key generation algorithm. As before, we quantify over all preservation sets, meaning that security must hold even if the adversary knows $\mathcal{T}$.

Unfortunately, this definition proves too strong for most domain extension settings of interest. Roughly speaking, unless m is very close to d, $|\mathcal{T}|$ is small, and d is large, one can give simple SPRP adversaries successful against any construction. The following theorem captures the negative result.

Theorem 4 *Fix a domain extension setting $(F, \mathcal{D}, \mathcal{T}, \mathcal{M})$. Let* $\mathrm{DEC} = (KT, E)$ *be a domain-extended cipher that preserves $\mathcal{T}$. Let $d = |\mathcal{D}|$ and $m = |\mathcal{M}|$ and $t = |\mathcal{T}|$. Then we give an SPRP adversary $\mathcal{A}$ that makes $q \le t$ queries for which*

$$\mathbf{Adv}^{\mathrm{sprp}}_{E}(\mathcal{A}) = 1 - \frac{d! \cdot (m-q)!}{m! \cdot (d-q)!} \ .$$

Proof: Our adversary picks any size q subset of $\mathcal{T}$ and queries each point in the subset to its encryption oracle. If any of the resulting ciphertexts are in $\mathcal{N} = \mathcal{M} \setminus \mathcal{D}$ it outputs 0, because this violates the definition of preserving $\mathcal{T}$. The adversary thus knows its oracle is a random permutation. If all q queries are in $\mathcal{D}$ it outputs 1.

In the real world, it is obvious the adversary outputs 1 with probability 1. In the case where the adversary's oracle is a random permutation, we have to treat the possibility of all the queries to the encryption oracle landing in $\mathcal{D}$ by chance. If this happens, the adversary is fooled into thinking its oracle is an extended FPE even though it's actually a random permutation. The probability that the first query's ciphertext is in $\mathcal{D}$ is $\frac{d}{m}$. The probability that the next one is also in $\mathcal{D}$ is $\frac{d(d-1)}{m(m-1)}$, because there are $d-1$ remaining points in $\mathcal{D}$ and $m-1$ points left in m. We multiply the probability of the first query also being in $\mathcal{D}$ because this probability is conditioned on that also happening. If we carry out this argument to q queries, we'll get

$$\frac{d(d-1)\cdots(d-q)}{m(m-1)\cdots(m-q)} = \frac{d! \cdot (m-q)!}{m! \cdot (d-q)!} \ .$$

∎

SEPRP security. Given the negative result about SPRP security, we turn to weaker, but still meaningful, security notions. The first is a relaxation of SPRP in which we do not seek to hide from an adversary that an extension has taken place. For a domain extension setting $(F, \mathcal{D}, \mathcal{T}, \mathcal{M})$, an adversary $\mathcal{A}$ and domain-extended cipher $\mathrm{DEC} = (KT, E)$, the $\mathrm{SEPRP0}^{\mathcal{A}}_{\mathrm{DEC}}$ and $\mathrm{SEPRP1}^{\mathcal{A}}_{\mathrm{DEC}}$ games in Figure 8 capture what we call "indistinguishability from an extended random permutation". (The games are implicitly parameterized by the domain extension setting.) There $\mathrm{ExtPerm}(\mathcal{M}, \mathcal{T}, \pi)$ is the set of all possible permutations $\tilde{\pi}$ such that for all $X \in \mathcal{T}$ it is the case that $\tilde{\pi}(X) = \pi(X)$. An adversary $\mathcal{A}$'s SEPRP advantage against DEC is defined as

$$\mathbf{Adv}^{\mathrm{seprp}}_{\mathrm{DEC}}(\mathcal{A}) = \left|\Pr\left[\,\mathrm{SEPRP1}^{\mathcal{A}}_{\mathrm{DEC}} \Rightarrow 1\,\right] - \Pr\left[\,\mathrm{SEPRP0}^{\mathcal{A}}_{\mathrm{DEC}} \Rightarrow 1\,\right]\right| .$$

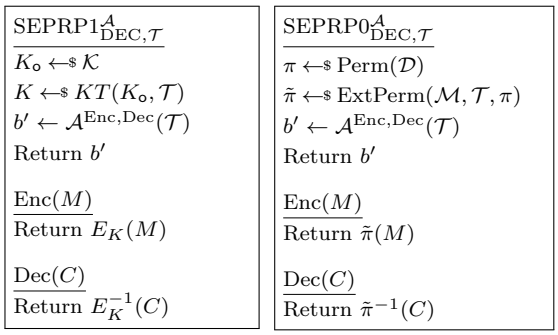

Figure 8: Games defining SEPRP security.

Zig-Zag for domain extension. Fixing a setting $(F, \mathcal{D}, \mathcal{T}, \mathcal{M})$, observe that the construction $\mathrm{ZZ} = (KT^{zz}, E^{zz})$ described in Figure 4 provides a domain-extended cipher with helper cipher $\overline{E}$. There is one subtlety in adapting ZZ to the domain extension setting: since $\overline{E}$ may output points in $\mathcal{M} \setminus \mathcal{D}$, the input to F^{-1} or F in the while loops of encryption and decryption may be outside $\mathcal{D}$. This can be fixed by a simple check which only runs F^{-1} or F if the input is in $\mathcal{D}$. Adding this branch does not change the behavior of the algorithm, since no point outside $\mathcal{D}$ can be the image under F of any point of $\mathcal{T}$. In our analyses below, this check will be implicit.

The next theorem captures its SEPRP security. Since this proof is substantially similar to the earlier proof of Theorem 3, we defer it to Appendix C. Here, as before, we do not analyze the query complexity of the reductions and use the worst-case upper bound of $q + |\mathcal{T}|$ queries.

Theorem 5 *Fix a domain-extension setting $(F, \mathcal{D}, \mathcal{T}, \mathcal{M})$ and let $\mathrm{ZZ} = (KT^{zz}, E^{zz})$ be the Zig-Zag domain-extended cipher using helper cipher $\overline{E}$. Let $\mathcal{A}$ be an SEPRP adversary making at most q queries to its oracles. Then the proof gives explicit adversaries $\mathcal{B}$ and $\mathcal{C}$ for which*

$$\mathbf{Adv}^{seprp}_{E^{zz}}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{sprp}}_{F}(\mathcal{B}) + \mathbf{Adv}^{\mathrm{sprp}}_{\overline{E}}(\mathcal{C}) \ .$$

The adversaries $\mathcal{B}, \mathcal{C}$ each run in time at most that of $\mathcal{A}$ plus at most a $\mathcal{O}(q) + |\mathcal{T}|$ overhead and each make at most $q + |\mathcal{T}|$ queries.

7 Understanding SEPRP Security

In this section we study SEPRP security in more detail, in particular understanding its relationship with prior security definitions. We'll explore the relationship between SEPRP and the notions of message recovery and message privacy security for ciphers introduced by Bellare et al. [3]. Throughout this section we fix a domain extension setting $(F, \mathcal{D}, \mathcal{T}, \mathcal{M})$, which is known to the adversary.

7.1 Message Recovery Security

The weakest definition from [3] is message-recovery security. At a high level it states that an attacker, given the encryption of some unknown message, should not be able to recover that message with probability better than that achieved by a simulator given no ciphertext. The adversary is additionally given an encryption oracle to which it can submit queries; the simulator is given access to an equality oracle that checks if the submitted message equals the target one. This latter reflects the fact that chosen-message attacks against an FPE scheme can always rule out messages one at a time by obtaining encryptions and comparing them to the challenge ciphertext.

We'll present a generalization of the standard message recovery definition called "generalized message recovery". We use the games gMR and gMRI to specify a simulation-style security target. The "real" game gMR tasks an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ with recovering a message chosen by $\mathcal{A}_1$ given its encryption under the domain-extended cipher DEC. We emphasize that $\mathcal{A}_1$ and $\mathcal{A}_2$ do not share any state (otherwise the definition would be vacuous). The adversary $\mathcal{A}_2$ has the ability to obtain encryptions on messages of its choosing. The "ideal" game gMRI tasks a simulator $\mathcal{S}$ to recover an identically distributed message X^* given some leakage $\mathrm{aux}(X^*)$ about it and the ability to query an equality oracle Eq that returns whether or not the submitted message equals X^*.

The generalized MR-advantage of an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ against a domain-extended cipher DEC is defined as

$$\mathbf{Adv}^{\mathrm{gmr}}_{\mathrm{DEC}}(\mathcal{A}, q', \mathrm{aux}) = \Pr\left[\, \mathrm{gMR}^{\mathcal{A}}_{\mathrm{DEC}} \Rightarrow \mathsf{true} \,\right] - \max_{\mathcal{S} \in \mathbb{S}_{q'}} \Pr\left[\, \mathrm{gMRI}^{\mathcal{A}_1, \mathcal{S}, \mathrm{aux}} \Rightarrow \mathsf{true} \,\right]$$

where the rightmost term is defined over $\mathbb{S}_{q'}$, the set of all simulators making at most q' queries to their Eq oracles. In what follows, the simulator can depend on an adversary $\mathcal{A}$. The string aux is the description of a function which takes a point of $\mathcal{M}$ and outputs either some information about it or $\perp$.

The value q' is a function of q, the number of queries the adversary makes in its experiment. Below, q' will be some small constant times q. When $q' > q$, it means that the security provided is weaker because the simulator needs more queries to its ideal functionality to achieve the same probability of success in its game. Intuitively, this means that the real oracle Enc leaks more information than the ideal oracle.

We will start by studying the relationship between MR security and SEPRP security. When $q' = q$ and aux is the function that always outputs $\perp$, our generalized MR definition above corresponds exactly to the message recovery definition from [3]. MR security was shown not to imply SPRP

$\underline{\text{gMR}_{\text{DEC}}}$	$\underline{\text{gMRI}}$	$\underline{\text{gMP}_{\text{DEC}}}$	$\underline{\text{gMPI}}$
$K_{\text{o}} \leftarrow\!\!\$\ \mathcal{K}$	$X^* \leftarrow\!\!\$\ \mathcal{A}_1(\mathcal{T})$	$K_{\text{o}} \leftarrow\!\!\$\ \mathcal{K}$	$X^* \leftarrow\!\!\$\ \mathcal{A}_1(\mathcal{T})$
$K \leftarrow\!\!\$\ KT(K_{\text{o}}, \mathcal{T})$	$X \leftarrow \mathcal{S}^{\text{Eq}}(\mathcal{T}, \text{aux}(X^*))$	$K \leftarrow\!\!\$\ KT(K_{\text{o}}, \mathcal{T})$	$X \leftarrow \mathcal{S}^{\text{Eq}}(\mathcal{T}, \text{aux}(X^*))$
$X^* \leftarrow\!\!\$\ \mathcal{A}_1(\mathcal{T})$	Return $(X = X^*)$	$X^* \leftarrow\!\!\$\ \mathcal{A}_1(\mathcal{T})$	Return $(X = \mathcal{A}_3(X^*))$
$Y^* \leftarrow E_K(X^*)$		$Y^* \leftarrow E_K(X^*)$	
$X \leftarrow \mathcal{A}_2^{\text{Enc}}(\mathcal{T}, Y^*)$		$Z \leftarrow \mathcal{A}_2^{\text{Enc}}(\mathcal{T}, Y^*)$	
Return $(X = X^*)$		Return $(Z = \mathcal{A}_3(X^*))$	
$\underline{\text{Enc(X)}}$	$\underline{\text{Eq(X)}}$	$\underline{\text{Enc(X)}}$	$\underline{\text{Eq(X)}}$
Return $E_K(X)$	Return $(X = X^*)$	Return $E_K(X)$	Return $(X = X^*)$

Figure 9: Generalized message-recovery and message privacy games.

security in [3], and it does not imply SEPRP security either. To demonstrate this, imagine we take an MR-secure cipher E over a size-d domain and add one bit to its domain, making it $d+1$ bits. Define a new cipher $E'(X)$ on this domain by calling E on the first d bits of X and concatenating the $d+1^{\text{st}}$ bit (in the clear) to make the ciphertext of X under E'. The MR-security of E' is reducible to the MR-security of E by a simple argument. However, this new cipher E' does not meet SEPRP security, because (with M and $E'(M)$ interpreted as integers) the quantity $|M - E'(M)|$ is the same whether the top bit of M is 1 or 0.

We can also show that SEPRP does not imply MR security. Take a similar setting in which the new domain $\mathcal{M}$ has $|\mathcal{M}| = m = 2 \cdot d$ where $|\mathcal{D}| = d$ and every point in $\mathcal{D}$ is preserved. We claim that for an SEPRP E, $\mathbf{Adv}_E^{\text{gmr}}(\mathcal{A}, q, \mathbb{B}) \geq \frac{1}{2}$, where $\mathbb{B}$ is the function that always outputs $\perp$ (meaning no information is leaked). To see this, take $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ and have $\mathcal{A}_2$ first check if the point it was given (i.e. the encryption of the point chosen by $\mathcal{A}_1$) is in $\mathcal{D}$. If so, it queries every point in $\mathcal{D}$ until it finds the right one. Likewise for $\mathcal{N}$. $\mathcal{A}$ wins the gMR game with probability 1, but any simulator wins the gMRI game with probability at most $\frac{1}{2}$ because it doesn't receive any information about the hidden point and only has q queries.

This is troubling, because we seem to have a separation in two directions when $q = q'$: generalized MR does not imply SEPRP, and SEPRP does not imply generalized MR. However, we can prove that SEPRP does imply generalized MR when the simulator is given some auxiliary information about the hidden point, namely whether or not it is in $\mathcal{T}$.

Theorem 6 *Fix a domain-extension setting $(F, \mathcal{D}, \mathcal{T}, \mathcal{M})$. For any domain-extended cipher* DEC *and adversary $\mathcal{A}$ making q oracle queries, we give in the proof an adversary $\mathcal{B}$ making q oracle queries and running in the same amount of time as $\mathcal{A}$ such that*

$$\mathbf{Adv}_{\text{DEC}}^{\text{gmr}}(\mathcal{A}, q, \text{aux}) \leq \mathbf{Adv}_{\text{DEC}}^{\text{seprp}}(\mathcal{B})$$

where $\text{aux}(X)$ *returns 1 if $X \in \mathcal{T}$ and 0 otherwise, for all $X \in \mathcal{M}$.*

Proof: Our adversary $\mathcal{B}$ is given the description of $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$. It runs $\mathcal{A}_1(\mathcal{T})$ and gets a point X^*, then runs $\mathcal{A}_2(\mathcal{T}, \text{Enc}(X^*))$, simulating $\mathcal{A}_2$'s Enc oracle using its own encryption oracle for the SEPRP game. When $\mathcal{A}_2$ outputs its guess X, $\mathcal{B}$ returns 1 if $(X = X^*)$ and 0 otherwise. By construction

$$\Pr\left[\,\text{gMR}_{\text{DEC}}^{\mathcal{A}} \Rightarrow \text{True}\,\right] = \Pr\left[\,\text{SEPRP1}_{\text{DEC}}^{\mathcal{B}} = 1\,\right].$$

To complete the proof we must show that

$$\max_{\mathcal{S} \in \mathbb{S}_q} \Pr\left[\,\text{gMRI}^{\mathcal{A}_1, \mathcal{S}, \text{aux}} \Rightarrow \text{true}\,\right] \geq \Pr\left[\,\text{SEPRP0}_{\text{DEC}}^{\mathcal{B}} = 1\,\right].$$

Construct a simulator $\mathcal{S}$ by giving it the target set $\mathcal{T}$ and the leakage bit that indicates whether the hidden point is in $\mathcal{T}$. $\mathcal{S}$ runs $\mathcal{A}_2(\mathcal{T}, X')$, where X' is a random point of $\mathcal{D}$ if its leakage bit indicates that the hidden point is preserved, and a random point of $\mathcal{M}$ otherwise. $\mathcal{S}$ simulates $\mathcal{A}$'s Enc oracle by taking each of $\mathcal{A}_2$'s queries and checking it against its own Eq oracle. If the Eq oracle returns true, $\mathcal{S}$ returns X'. Otherwise, $\mathcal{S}$ returns a random (subject to permutivity) point of $\mathcal{D}$ if $\mathcal{A}_2$'s query is in $\mathcal{T}$, and a random point of $\mathcal{M}$ otherwise. $\mathcal{S}$ returns whatever $\mathcal{A}_2$ does. By inspection, $\mathcal{S}$ is simulating the same environment for $\mathcal{A}_2$ as $\mathcal{B}$ does in the ideal SEPRP game, because in either case the environment is lazy-sampling a random ideal extended permutation. Thus, the probability of $\mathcal{S}$ winning is exactly the probability of $\mathcal{B}$ guessing 1 in the SEPRP0 game. The max value of the left-hand side is at least the success probability of this simulator, so the inequality holds. ∎

We can also prove the following relationship between different parameterizations of the generalized MR games. Intuitively, this theorem says that leaking whether the hidden point is in $\mathcal{T}$ is equivalent to speeding up a guessing attack by a factor of two.

Theorem 7 *Fix a domain-extension setting $(F, \mathcal{D}, \mathcal{T}, \mathcal{M})$. For any domain-extended cipher* DEC *and adversary $\mathcal{A}$ making q queries,*

$$\mathbf{Adv}^{\mathrm{gmr}}_{\mathrm{DEC}}(\mathcal{A}, 2q, \mathbb{B}) \leq \mathbf{Adv}^{\mathrm{gmr}}_{\mathrm{DEC}}(\mathcal{A}, q, \mathrm{aux})$$

where aux *and* $\mathbb{B}$ *are as above.*

Proof: First, observe that

$$\Pr\left[\,\mathrm{gMR}^{\mathcal{A}}_{\mathrm{DEC}} \Rightarrow \mathrm{true}\,\right] = \Pr\left[\,\mathrm{gMR}^{\mathcal{A}}_{\mathrm{DEC}} \Rightarrow \mathrm{true}\,\right].$$

This is tautological because the gMR game is the same in either case; only the gMRI game changes.

To complete the proof we need to show that

$$\max_{\mathcal{S} \in \mathbb{S}_{2q+2}} \Pr\left[\,\mathrm{gMRI}^{\mathcal{A}_1, \mathcal{S}, \mathbb{B}} \Rightarrow \mathrm{true}\,\right] \geq \max_{\mathcal{S}' \in \mathbb{S}_q} \Pr\left[\,\mathrm{gMRI}^{\mathcal{A}_1, \mathcal{S}', \mathrm{aux}} \Rightarrow \mathrm{true}\,\right].$$

The simulator $\mathcal{S}$ is given a description of the $\mathcal{S}'$ that maximizes the right-hand side and runs it twice — once with the leakage bit set to 0 and once with the bit set to 1. $\mathcal{S}$ answers $\mathcal{S}'$'s Eq queries with its own Eq oracle. If one of $\mathcal{S}'$'s guesses is correct, $\mathcal{S}$ returns that as its guess. Since $\mathcal{S}$ runs $\mathcal{S}'$ with the leakage bit set to both possible values, if $\mathcal{S}'$ wins in either case, $\mathcal{S}$ wins as well. Thus, the success probability of $\mathcal{S}$ is at least the success probability of $\mathcal{S}'$. ∎

7.2 Message Privacy

In [3] a (strictly stronger) definition than message recovery is proposed. They refer to this definition as message privacy. It says, roughly, that no adversary can compute any function of the message given only its ciphertext. Message-recovery security is a special case of message privacy where the function the adversary wants to compute is the equality function. We define the generalized MP-advantage of an adversary $\mathcal{A}$, using the games gMP and gMPI in Figure 9, as

$$\mathbf{Adv}^{\mathrm{gmp}}_{E}(\mathcal{A}, q', \mathrm{aux}) = \Pr\left[\,\mathrm{gMP}^{\mathcal{A}}_{\mathrm{DEC}} \Rightarrow \mathsf{true}\,\right] - \max_{\mathcal{S} \in \mathbb{S}_{q'}} \Pr\left[\,\mathrm{gMPI}^{\mathcal{S}, \mathrm{aux}} \Rightarrow \mathsf{true}\,\right].$$

We will use this generalized definition instead of the one used in [3] because extended permutations leak more information to adversaries than standard SPRPs. To demonstrate the necessity of this generalized definition, we'll prove that SEPRP security does not imply the standard message privacy definition from [3], which corresponds to our generalized definition when $q = q'$ and $\mathrm{aux} = \perp$.

Theorem 8 *Fix a domain-extension setting $(F, \mathcal{D}, \mathcal{T}, \mathcal{M})$ for which $\mathcal{T} = \mathcal{D}$, i.e., every point is preserved. For any domain-extended cipher* DEC*, the proof gives a specific adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$ in the message privacy game such that*

$$\mathbf{Adv}^{\mathrm{gmp}}_{\mathrm{DEC}}(\mathcal{A}, 0, \perp) = 1 - \max\left(\frac{d}{m}, \frac{n}{m}\right) .$$

Proof: The algorithm $\mathcal{A}_1$ samples uniformly from its input space. The function represented by $\mathcal{A}_3$ is

$$\mathcal{A}_3(m) = \begin{cases} 1 & \text{if } m \in \mathcal{D} \\ 0 & \text{otherwise} \end{cases} .$$

Our adversary $\mathcal{A}$ wins with probability 1 by checking whether the point Y^* it is given is in $\mathcal{D}$ or $\mathcal{M} \setminus \mathcal{D}$. Because every point is preserved, $\mathcal{A}$ always computes the function correctly. The simulator $\mathcal{S}$ does not get any Eq queries because $\mathcal{A}$ used no encryption queries, and its auxiliary function always outputs $\perp$, so the simulator's optimal strategy is to output 1 if $d > n$ and 0 otherwise. A point from the larger of the two sets is more likely, so the simulator wins with probability $\max\left(\frac{d}{m}, \frac{n}{m}\right)$. ∎

We can, however, prove that SEPRP does imply generalized message privacy when an oracle for membership in $\mathcal{T}$ is given to the simulator.

Theorem 9 *Fix a domain-extension setting $(F, \mathcal{D}, \mathcal{T}, \mathcal{M})$. For any domain-extended cipher* DEC *and adversary $\mathcal{A}$ making q oracle queries, we give in the proof an adversary $\mathcal{B}$ making q queries and running in the same amount of time as $\mathcal{A}$ such that*

$$\mathbf{Adv}^{\mathrm{gmp}}_{\mathrm{DEC}}(\mathcal{A}, q, \mathrm{aux}) \leq \mathbf{Adv}^{\mathrm{seprp}}_{\mathrm{DEC}}(\mathcal{B})$$

where aux *returns 1 if its input is in $\mathcal{T}$ and 0 otherwise.*

Proof: Our adversary $\mathcal{B}$ is given the description of $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$. It runs $\mathcal{A}_1(\mathcal{T})$ and gets a point X^*, then runs $\mathcal{A}_2(\mathrm{Enc}(X^*))$, simulating $\mathcal{A}_2$'s Enc oracle using its own encryption oracle for the SEPRP game. When $\mathcal{A}_2$ outputs its guess Z, $\mathcal{B}$ returns 1 if $(Z = \mathcal{A}_3(X^*))$ and 0 otherwise. By construction

$$\Pr\left[\,\mathrm{gMP}^{\mathcal{A}}_{\mathrm{DEC}} \Rightarrow \mathrm{True}\,\right] = \Pr\left[\,\mathrm{SEPRP1}^{\mathcal{B}}_{\mathrm{DEC}} = 1\,\right] .$$

It suffices to show that

$$\max_{\mathcal{S} \in \mathbb{S}_q} \Pr\left[\,\mathrm{gMPI}^{\mathcal{S},\mathrm{aux}} \Rightarrow \mathsf{true}\,\right] \geq \Pr\left[\,\mathrm{SEPRP0}^{\mathcal{B}}_{\mathrm{DEC}} = 1\,\right] .$$

Define a simulator $\mathcal{S}$ that takes $\mathcal{T}$ and the value of $\mathrm{aux}(X^*)$. The simulator $\mathcal{S}$ runs $\mathcal{A}_2(X')$ where X' is a random point of $\mathcal{D}$ if the simulator's leakage from aux indicates the hidden point is in $\mathcal{T}$ and X' is a random point of $\mathcal{M}$ otherwise. The simulator simulates $\mathcal{A}_2$'s Enc oracle by first using its own Eq oracle to check if $\mathcal{A}_2$'s guess is equal to the hidden point. If $\mathcal{S}$'s oracle returns true, $\mathcal{S}$ returns X' in response to $\mathcal{A}_2$'s query. If it returns false, $\mathcal{S}$ checks if the queried point is in $\mathcal{T}$. If it is, $\mathcal{S}$ returns a random point of $\mathcal{D}$, else $\mathcal{S}$ returns a random point of $\mathcal{M}$. The simulator makes both choices subject to permutivity. When $\mathcal{A}_2$ returns its guess for $\mathcal{A}_3(X^*)$, $\mathcal{S}$ outputs the same guess. Since $\mathcal{S}$ is simulating the same environment for $\mathcal{A}_2$ as $\mathcal{B}$ does in the case where $\mathcal{B}$'s oracle is an ideal extended permutation, the probability of this simulator $\mathcal{S}$ winning is exactly the probability of $\mathcal{B}$ guessing 1 in the SEPRP0 game. The true max of the left-hand side is at least the probability that this $\mathcal{S}$ we've constructed wins the gMPI game, so the inequality holds. ∎

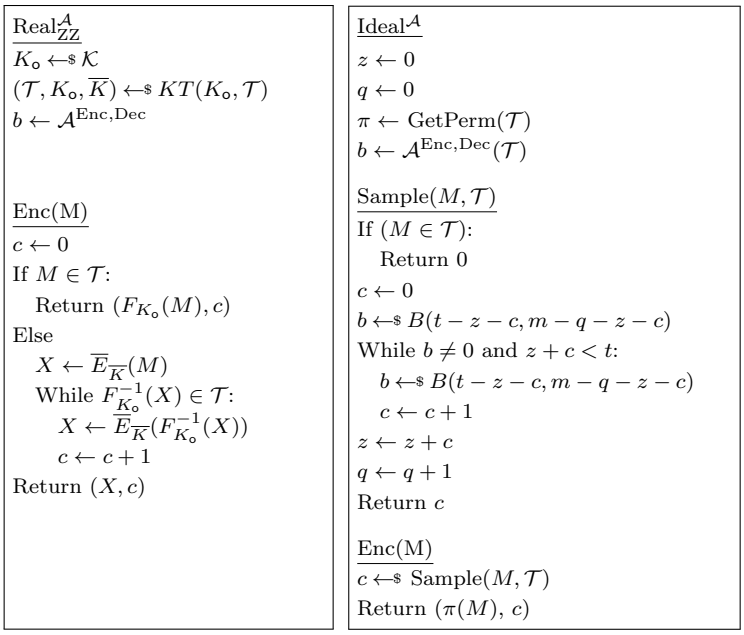

Figure 10: Games for Theorem 10. In the right-hand game $t = |\mathcal{T}|$. We have omitted the code of the decryption oracles; they are implemented in the obvious way.

8 The Zig-Zag Construction and Side-Channel Resistance

One important question when designing any encryption scheme that contains branches on secret data or has variable timing for different points is whether this gives rise to any kind of side-channel attack. Timing side-channels have proven particularly dangerous in applications [1,2,6,8,15,16,19] so we would like to prove the Zig-Zag construction at least does not give rise to a timing side-channel. In this section, we will prove that the time taken to encrypt or decrypt with the Zig-Zag construction does not leak useful information to an adversary about the encrypted message.

Fix some domain completion[1] setting $(F, \mathcal{M}, \mathcal{T})$ and let $\mathrm{ZZ} = (KT^{zz}, E^{zz})$ be the Zig-Zag construction for it. We define two games, detailed in Figure 10. The first, Real$^{\mathcal{A}}_{\mathrm{ZZ}}$, gives the adversary $\mathcal{A}$ access to a Zig-Zag enciphering oracle that additionally reveals the number of iterations of the inner loop of Zig-Zag. The second, Ideal$^{\mathcal{A}}$ gives the adversary $\mathcal{A}$ access to an oracle that returns a random permutation applied to the message as well as a simulated while-loop count that only uses whether $M \in \mathcal{T}$. In the Sample procedure, the function $B(x, y)$ generates a random bit that is 1 with probability $\frac{x}{y}$. This is used to simulate the timing information available to $\mathcal{A}$ in the real game. Intuitively, the distinguishing advantage between these two games lets us measure how much additional distinguishing advantage an adversary has if it knows how many iterations of Zig-Zag are required to encrypt or decrypt a point. Fortunately, the following theorem proves this information gives no additional advantage. Here, as before, we use the worst-case query complexity of $q + |\mathcal{T}|$ for simplicity in the reduction.

Theorem 10 *Fix a domain completion setting $(F, \mathcal{M}, \mathcal{T})$ and let $\mathrm{ZZ} = (KT^{zz}, E^{zz})$ be the Zig-Zag cipher for it built using an underlying helper cipher $\overline{E}$. Then for any $\mathcal{A}$ making at most q queries the proof gives specific adversaries $\mathcal{B}$ and $\mathcal{C}$ such that*

$$\left|\Pr\left[\, \mathit{Real}^{\mathcal{A}}_{\mathrm{ZZ}} \Rightarrow 1 \,\right] - \Pr\left[\, \mathit{Ideal}^{\mathcal{A}} \Rightarrow 1 \,\right]\right| \leq \mathbf{Adv}^{\mathrm{sprp}}_{\overline{E}}(\mathcal{B}) + \mathbf{Adv}^{\mathrm{sprp}}_{F}(\mathcal{C})\ .$$

[1] A result similar to Theorem 10 holds for domain extension and the proof is largely identical, so we will omit it.

Adversaries $\mathcal{B}, \mathcal{C}$ each run in time at most that of $\mathcal{A}$ plus at most a $\mathcal{O}(q) + |\mathcal{T}|$ overhead and each make at most $q + |\mathcal{T}|$ queries.

We defer the proof to Appendix B. This theorem lets us say with certainty that information on how long encryption of a particular point takes leaks only whether it is in $\mathcal{T}$ or not. Since in a chosen-plaintext attack the adversary already knows whether $M \in \mathcal{T}$, this means the adversary learns nothing. Intuitively this is because, for every point $M \notin \mathcal{T}$ the distribution of M's zig-zag lengths is the same.

9 Domain Extension when Adversaries Do Not Know $\mathcal{T}$

In Section 6 we demonstrated that we cannot achieve SPRP security while preserving a subset of the original domain if the adversary knows which subset is preserved. We ask, then, if there are weaker adversarial settings in which SPRP-like security can be attained. In particular, we may want to know what the strongest "weaker" adversary is — namely, how much information can we reveal about $\mathcal{T}$ before SPRP becomes provably impossible. In this section we provide a constructive partial answer to this question by building an SPRP-secure scheme in the setting where the adversary only knows $|\mathcal{T}| = t$, the size of the preserved set, but does not know which elements it contains. This weakening of the adversary is motivated not only by theoretical questions, but by practical settings in which the attacker, through application logs or other information, is able to infer the number of ciphertexts in the database before an extension has occurred.

In terms of security goal, we target SPRP security in a setting where the adversary knows $\mathcal{D}$ but the preserved set is chosen uniformly from the subsets of size t of the original domain $\mathcal{D}$, and the random coins used to make this choice are hidden from the adversary. We leave treating other cases as an open question.

Observe that if $t > d - n$, a random permutation has a nonzero probability of having fewer than t elements of $\mathcal{D}$ mapped into $\mathcal{D}$. When a permutation maps a point of $\mathcal{D}$ back to $\mathcal{D}$, we say that it "domain-preserves" that point. Since one of our goals for domain extension is to preserve mappings for points in $\mathcal{T}$ (which are domain-preserved mappings under the extended permutation) there must be at least t domain-preserved points in our permutation. To make this more intuitive, consider how few points can possibly be domain-preserved in any permutation. This occurs when (for $d > n$) as many points as possible are mapped from $\mathcal{D}$ to $\mathcal{N}$. Since this is a permutation, only n points can have ciphertexts in $\mathcal{N}$. The rest *have* to be domain-preserved. If this happens, n points of $\mathcal{D}$ are not domain-preserved, so $d - n$ points are. If t is indeed greater than this strict lower bound, any DEC that preserves $\mathcal{T}$ cannot include some possible permutations (i.e., the ones that domain-preserve between $t - 1$ and $d - n$ points). This will give a distinguishing advantage to an adversary.

The Recursive Zig-Zag. For the case that $t \leq d - n$, any permutation on $\mathcal{M}$ domain-preserves at least t elements of $\mathcal{D}$. We will use this fact to construct a Zig-Zag algorithm that achieves SPRP security for domain extension. Since the key transformation acts in a recursive fashion on its state, we will call the algorithm the "Recursive Zig-Zag" (RZZ). As in our other constructions a helper cipher $\overline{E}$ will be used. The key transformation works by selecting a set of points that are domain-preserved under the helper cipher and, for each point τ in $\mathcal{T}$, "swapping" the image of one of these points with the image of τ if τ is *not* domain-preserved. This is done so the number of domain-preserved points in the resulting permutation is unchanged. Points that are swapped are stored in a lookup table Emap. Below, we will prove SPRP security of the RZZ and demonstrate that the expected amortized cost of the key transform KT^{rzz} is constant for each point of $\mathcal{T}$.

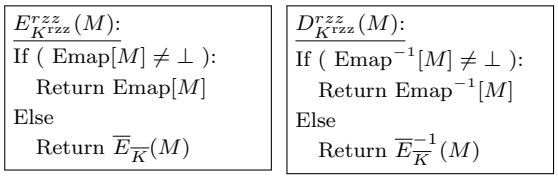

Figure 11: Recursive Zig-Zag encryption and decryption algorithms. The key $K^{\text{rzz}} = \langle \text{Emap}, \overline{K} \rangle$ contains both the lookup table Emap and the key $\overline{K}$ for $\overline{E}$.

To motivate the RZZ, it may be useful to give a concrete example of why the previous Zig-Zag construction cannot be SPRP-secure for domain extension when only t is known. Take $d = 99$, $t = 98$ and $n = 1$. In this case, Zig Zag will have a 50% probability that the newly added element maps to itself. However, the probability of that happening in a random permutation is 1%. The main cause of the problem is that standard Zig-Zag may change the size of the domain-preserved set. In KT^{rzz} we guarantee that does not happen.

The construction. Figure 11 shows the encryption and decryption algorithms. Encryption consults the lookup table Emap for the message's associated output, returning it if found. Otherwise it returns the value of the helper cipher $\overline{E}$. Decryption works the same way. The new key $K^{\text{rzz}} = \langle \text{Emap}, \overline{K} \rangle$ output by KT^{rzz} contains the lookup table Emap which is precomputed by the key transformation algorithm in Figure 12. We use the notation $\text{Emap}[x]$ to refer to the mapping of the element x under Emap. The notation $\text{Emap}^{-1}[y]$ returns the value x such that $\text{Emap}[x] = y$. If Emap does not provide a mapping for x, the value of $\text{Emap}[x]$ is $\perp$. If there is no point mapped to y in Emap, $\text{Emap}^{-1}[y]$ will likewise output $\perp$.

The KT^{rzz} algorithm. The key transformation algorithm records all the values modified during the t iterations. At the start of the ith iteration of KT^{rzz}, Emap contains the values changed in all previous iterations. We begin the iteration by computing $\tau_{\text{old}} \leftarrow E^{rzz}_{K^{\text{rzz}}}(\tau_i)$ where $E^{rzz}_{K^{\text{rzz}}}$ is computed as in Figure 11. There are then three cases. We will explain each in turn, referring to the case numbers given in Figure 12.

Case 1 ($\tau_{\text{old}} = F_{K_\circ}(\tau_i)$): This occurs if $E^{rzz}_{K^{\text{rzz}}}$ already contains the correct mapping for τ_i. That would happen if the helper cipher $\overline{E}_{\overline{K}}$ maps τ_i to $F_{K_\circ}(\tau_i)$. We simply update Emap and continue.

Case 2 ($\tau_{\text{old}} \in \mathcal{D}$): This occurs if $E^{rzz}_{K^{\text{rzz}}}$ domain-preserves τ_i. Here we do not need to worry about biasing the number of domain-preserved points by preserving τ_i's mapping to $F_{K_\circ}(\tau_i)$ because both $F_{K_\circ}(\tau_i)$ and τ_{old} are in $\mathcal{D}$. In this case we can do a zig-zag as in the previous Zig-Zag construction, assigning τ_{old} to the decryption of $F_{K_\circ}(\tau_i)$ under $D^{rzz}_{K^{\text{rzz}}}$ and τ_i to $F_{K_\circ}(\tau_i)$, as desired.

Case 3 ($\tau_{\text{old}} \in \mathcal{N}$): This occurs if $E^{rzz}_{K^{\text{rzz}}}$ does *not* domain-preserve τ_i. This is the case that requires special handling, since if we patch $E^{rzz}_{K^{\text{rzz}}}$ to map τ_i to $F_{K_\circ}(\tau_i)$ we may increase the number of domain-preserved points of $E^{rzz}_{K^{\text{rzz}}}$ and give a distinguishing advantage to an adversary. We use rejection sampling on points of $\mathcal{D} \setminus \{\tau_1, \ldots, \tau_{i-1}\}$ to find a point not in $\mathcal{T}$ that is domain-preserved under $E^{rzz}_{K^{\text{rzz}}}$. Such a point is guaranteed to exist by our assumption that $t \leq d - n$. When we find such a point p_i, we record it with its new image τ_{old} in Emap. Finally, we assign its old image under $E^{rzz}_{K^{\text{rzz}}}$, p_i^{old}, to be the image of $D^{rzz}_{K^{\text{rzz}}}(F_{K_\circ}(\tau_i))$ to preserve permutivity. Once we select p_i it cannot be selected in a subsequent iteration, since it will no longer be domain-preserved under $E^{rzz}_{K^{\text{rzz}}}$.

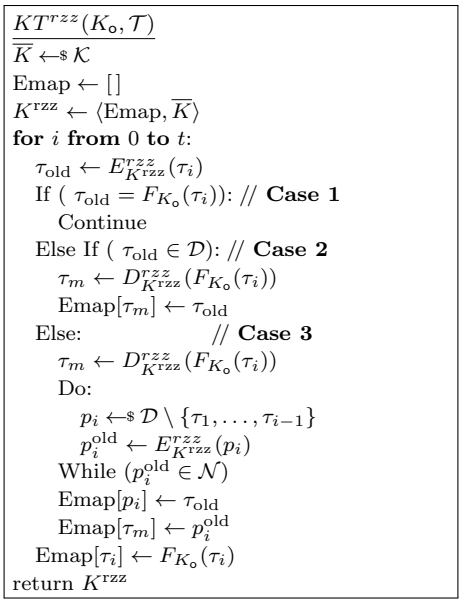

Figure 12: The Recursive Zig-Zag key transformation. The original cipher is F. The helper cipher is $\overline{E}$.

Note that we do not need special handling of the case that on the i^{th} iteration of KT^{rzz} we assign $E^{rzz}_{K^{rzz}}(\tau_i) = \tau_j$ for some $j > i$. We *will* change the value of $E^{rzz}_{K^{rzz}}(\tau_j)$ on the i^{th} iteration (violating correctness), but we will fix it in the j^{th} iteration.

The number of points changed in each of the t transformations is at most 3. Consequently, the number of points we need to pre-calculate is at most $3t$ and with the result of precomputation we need to encode at most $6t$ values—6 times as much as we need to encode to remember $\mathcal{T}$. Next, we will argue two correctness conditions for the RZZ.

Theorem 11 *Let $E^{rzz}_{K^{rzz}}$ be the function obtained after running KT^{rzz} on key K_o for F and preservation set $\mathcal{T}$. Then*

(1) $E^{rzz}_{K^{rzz}}$ is a permutation, and

(2) for $\tau_i \in \mathcal{T}$, $E^{rzz}_{K^{rzz}}(\tau_i) = F_{K_o}(\tau_i)$.

Proof: Let $E^{rzz}_{K_i^{rzz}}$ be the RZZ after the ith iteration of KT^{rzz}. We will proceed by induction on i. For $i = 0$ both correctness properties hold trivially. Next, assume that $E^{rzz}_{K_{i-1}^{rzz}}(\tau_j) = F(\tau_j)$ for all $j < i$, and that $E^{rzz}_{K_{i-1}^{rzz}}$ is a permutation. We now look at $E^{rzz}_{K_i^{rzz}}$. In all three cases above, $E^{rzz}_{K_i^{rzz}}(\tau_i) = F(\tau_i)$. To prove property (2) we need to show that $E^{rzz}_{K_i^{rzz}}(\tau_j) = F_{K_o}(\tau_j)$ for all $j \leq i$. If the ith iteration uses case 1, this is true trivially by the induction hypothesis. If it uses case 2, this is true for two reasons. One is that $\tau_{\text{old}} \neq F_{K_o}(\tau_j)$ for any $j < i$, by the permutivity of $E^{rzz}_{K_{i-1}^{rzz}}$ and the fact that all points in $\mathcal{T}$ are distinct. The other is that $\tau_m \neq \tau_j$ for any $j \leq i$, again by the permutivity of $E^{rzz}_{K_{i-1}^{rzz}}$. Thus, the assignment Emap[τ_m] $= \tau_{\text{old}}$ in case 2 does not change the mapping of τ_i or any previous τ_j. If it uses case 3, we use the permutivity of $E^{rzz}_{K_{i-1}^{rzz}}$ to argue that p_i and p_i^{old} are distinct from any previous τ_j or $F_{K_o}(\tau_j)$. In case 3, $\tau_{\text{old}} \in \mathcal{N}$, so τ_i cannot be chosen as p_i either. The correctness of τ_m and τ_{old} argued above still hold in case 3. Hence, for all $j \leq i$ we have $E^{rzz}_{K_i^{rzz}}(\tau_j) = F(\tau_j)$.

Next, we need to show that if $E^{rzz}_{K_{i-1}^{rzz}}$ is a permutation then $E^{rzz}_{K_i^{rzz}}$ is also a permutation. If the ith iteration uses case 1, this is trivially true. We argued above that τ_{old} is distinct from $F_{K_o}(\tau_j)$

and τ_m is distinct from τ_j for all $j < i$, which suffices to show $E^{rzz}_{K_i^{\text{rzz}}}$ is a permutation if it uses case 2. Note that we *cannot* sample the same p twice in case 3. The set of points sampled in any previous iterations going through case 3 in KT^{rzz} is excluded from consideration because we give those points an image in $\mathcal{N}$. Above, we argued that p_i and p_i^{old} are distinct from any previous τ_j or $F_{K_\circ}(\tau_j)$. The point p_i *can* be equal to τ_m for some previous iteration, but this does not violate permutivity, since we simply change its image in Emap to p_i^{old} and the mapping of p_i to τ_{old}. Thus, in all cases $E^{rzz}_{K_i^{\text{rzz}}}$ is a permutation, and property (1) holds.

In Theorem 13, we show that $E^{rzz}_{K^{\text{rzz}}}$ is an SPRP. In the course of that proof, we show the runtime of KT^{rzz} is $\mathcal{O}(t)$ except with exponentially small (in t) probability. Intuitively, this is because in expectation case 3 terminates after a constant number of iterations, and case 3 happens at most t times.

9.1 Security of the construction

We will now state the theorems showing the cipher $E^{rzz}_{K^{\text{rzz}}}$ with key K^{rzz} output by KT^{rzz}, is an SPRP. First, we assert an information-theoretic bound, Theorem 12. The proof appears in Appendix D. Intuitively, this theorem shows that if $\overline{E}$ and F are uniformly random permutations, the permutation $E^{rzz}_{K^{\text{rzz}}}$ resulting from constructing Emap from F and $\overline{E}$ as in KT^{rzz} is also uniformly random. Thus, no adversary (even unbounded) can distinguish the $E^{rzz}_{K^{\text{rzz}}}$ from a uniformly random permutation on $\mathcal{M}$. We will complete the proof that $E^{rzz}_{K^{\text{rzz}}}$ is an SPRP with two complexity-theoretic steps.

Theorem 12 *Let $\mathcal{T}$ be a uniformly random subset of $\mathcal{D}$, $|\mathcal{T}| = t$, and $t \leq d - n$. Let $E^{rzz}_{K^{\text{rzz}}}$ be a random variable denoting the permutation over $\mathcal{M}$ induced by the RZZ algorithm run with the output of KT^{rzz}, instantiated with $\overline{E}$ and F as uniformly random permutations over $\mathcal{M}$ and $\mathcal{D}$, respectively. Let $|\mathcal{M}| = m$. For any fixed permutation Π on $\mathcal{M}$,*

$$\Pr\left[\, E^{rzz}_{K^{\text{rzz}}} = \Pi \,\right] = \frac{1}{m!} \,.$$

Theorem 13 *Assume that $\overline{E}$ and F are ciphers on domains $\mathcal{M}$ and $\mathcal{D}$, respectively. Let $\mathcal{T}$ be a uniformly random subset of $\mathcal{D}$, $|\mathcal{T}| = t$, and $t \leq d - n$. Let $|\mathcal{D}| = d$, $|\mathcal{N}| = n$, and $c = \frac{n}{d}$. Let $\mathcal{A}$ be an SPRP adversary against $E^{rzz}_{K^{\text{rzz}}}$ which receives t as input and makes at most q queries to its oracles. Then the proof gives an adversary $\mathcal{B}$ and an adversary $\mathcal{C}$ such that*

$$\mathbf{Adv}^{\text{sprp}}_{E^{rzz}_{K^{\text{rzz}}}}(\mathcal{A}(t)) \leq \mathbf{Adv}^{\text{sprp}}_{F}(\mathcal{B}) + \mathbf{Adv}^{\text{sprp}}_{\overline{E}}(\mathcal{C}) + e^{-(49(1-c)^2 t)/4} \,.$$

Both $\mathcal{B}$ and $\mathcal{C}$ run in time at most that of $\mathcal{A}$ plus a $\mathcal{O}(q + |\mathcal{T}|)$ overhead. Adversary $\mathcal{B}$ makes at most t queries. Adversary $\mathcal{C}$ makes at most $q + 8t$ queries.

Proof: We need to give two reductions ($\mathcal{B}$ and $\mathcal{C}$) to prove this result. The reduction $\mathcal{B}$ is straightforward, but $\mathcal{C}$ is slightly complicated because KT^{rzz} uses rejection sampling to find domain-preserved points, which means in the worst case it could require $\mathcal{N}$ queries to complete KT^{rzz}. We must disallow this worst-case behavior if we want $\mathcal{C}$ to be efficient.

The code of $\mathcal{C}$ is in Figure 13. It simulates $\mathcal{A}$'s environment using its oracle in place of calls to $\overline{E}$ in E^{rzz}, D^{rzz}, and KT^{rzz}. It samples $\mathcal{T}$ uniformly at random from $\mathcal{D}$ and runs $F_{K_\circ}$ on each element to give $\mathsf{Range}_{\mathcal{T}}$. Once it completes KT^{rzz}, each of $\mathcal{A}$'s queries can be answered with at most one query to $\mathcal{C}$'s oracle. Thus, to bound the number of queries $\mathcal{C}$ makes we must bound the number of queries it uses in KT^{rzz}. To do this, our $\mathcal{C}$ keeps track of the number of samples it makes. This

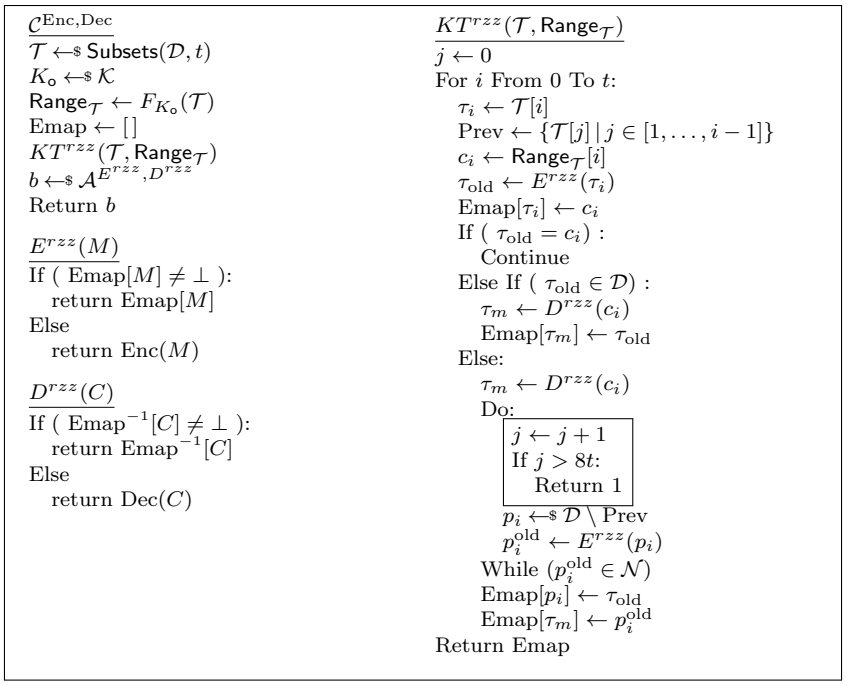

Figure 13: Code for adversary $\mathcal{C}$ of Theorem 13. The function $\mathsf{Subsets}(A, B)$ samples a size-B subset from the set A uniformly at random. The notation $F_{K_{\mathsf{o}}}(\mathcal{T})$ refers to the set obtained by running F on each element of $\mathcal{T}$.

is the purpose of the boxed code in Figure 13. (The code is boxed to highlight its importance, not to indicate inclusion or exclusion in the game.) If $\mathcal{C}$ samples more than $8t$ points during KT^{rzz} it halts and outputs 1. When $\mathcal{C}$ is in game SPRP1, it will correctly output 1 if the number of samples exceeds $8t$. Thus, to bound the distinguishing advantage we must bound the probability the number of samples exceeds $8t$ (i.e. that $j > 8t$) when $\mathcal{C}$'s oracle is a random permutation. We can view this as a hypergeometric experiment in which the domain-preserved points in $\mathcal{D}$ are "successes" and the other points are "failures". Our problem is to bound the probability of fewer than t "successes" in $8t$ samples. Note that by assumption there are at least $d - n$ "successes", i.e. domain-preserved points, and d total points. Viewing the counter j as a random variable over the choices made in calls to $\mathcal{C}$'s oracles in game SPRP0, we can see that $\mathrm{E}[\,j\,] = 8t\frac{d-n}{d}$. We rewrite the tail bound on the hypergeometric distribution given in Lemma 1 to obtain

$$\Pr\left[\, j < \left(\frac{d-n}{d} - v\right) 8t \,\right] \leq e^{-16v^2 t} \;.$$

Setting $v = \frac{7(d-n)}{8d}$ gives $j < t$. To simplify the bound we will rewrite $d - n$ to $(1-c)d$, which makes our term $v = \frac{7(1-c)}{8}$. Plugging this in and cancelling, our bound becomes $e^{-(49(1-c)^2 t)/4}$.

The reduction $\mathcal{B}$ is standard and lets us transition to a game in which F is a random permutation. Note that $\mathcal{B}$ simulates $\overline{E}$ by lazy-sampling a random permutation. This allows it to avoid rejection sampling by simply picking p_i^{old} to be domain-preserved in case 3. It therefore runs KT^{rzz} in time $O(t)$ in the worst case. Applying Theorem 12 completes the proof. ∎

Acknowledgments

The authors thank Terence Spies for suggesting this problem to them, and for discussions about how it impacts industry practice. The authors also thank the anonymous reviewer whose observations led to Section 4, and for his or her permission to include it. This work was supported in part by NSF grants CNS-1514163, CNS-1330308, and CNS-1558500 as well as a generous gift by Microsoft.

References

[1] Onur Acıiçmez, Çetin Kaya Koç, and Jean-Pierre Seifert. On the power of simple branch prediction analysis. In *2nd ACM ASIACCS*, 2007.

[2] Onur Acıiçmez and Jean-Pierre Seifert. Cheap hardware parallelism implies cheap security. In *Fourth International Workshop on Fault Diagnosis and Tolerance in Cryptography*, 2007.

[3] Mihir Bellare, Thomas Ristenpart, Phillip Rogaway, and Till Stegers. Format-preserving encryption. In *Selected Areas in Cryptography*, 2009.

[4] Mihir Bellare and Phillip Rogaway. Code-based game-playing proofs and the security of triple encryption. *EUROCRYPT*, 2004.

[5] Mihir Bellare, Phillip Rogaway, and Terence Spies. The FFX mode of operation for format-preserving encryption. `http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/ffx/ffx-spec.pdf`.

[6] Daniel J Bernstein. Cache-timing attacks on AES, 2005. Preprint available at `http://cr.yp.to/papers.html#cachetiming`.

[7] John Black and Phillip Rogaway. Ciphers with arbitrary finite domains. In *CT-RSA 2002*. 2002.

[8] David Brumley and Dan Boneh. Remote timing attacks are practical. *Computer Networks*, 2005.

[9] V. Chavátal. The tail of the hypergeometric distribution. *Discrete Mathematics*, 1979.

[10] PCI Security Standards Council. The payment card industry data security standard specification. `https://www.pcisecuritystandards.org/security_standards/documents.php?agreements=pcidss&association=pcidss`.

[11] Kevin P. Dyer, Scott E. Coull, Thomas Ristenpart, and Thomas Shrimpton. Protocol misidentification made easy with format-transforming encryption. In *ACM CCS*, 2013.

[12] Protegrity Inc. Protegrity data protection methods. `http://www.protegrity.com/data-security-platform/#protection-methods`.

[13] Skyhigh Networks Inc. Skyhigh for Salesforce. `https://www.skyhighnetworks.com/product/salesforce-encryption/`.

[14] Moses Liskov, Ronald L Rivest, and David Wagner. Tweakable block ciphers. In *CRYPTO*. 2002.

[15] Fangfei Liu, Qian Ge, Yuval Yarom, Frank Mckeen, Carlos Rozas, Gernot Heiser, and Ruby B Lee. CATalyst: Defeating last-level cache side channel attacks in cloud computing. In *IEEE Symposium on High-Performance Computer Architecture*, 2016.

[16] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B Lee. Last-level cache side-channel attacks are practical. In *IEEE Symposium on Security and Privacy*, 2015.

[17] Daniel Luchaup, Kevin P. Dyer, Somesh Jha, Thomas Ristenpart, and Thomas Shrimpton. LibFTE: A user-friendly toolkit for constructing practical format-abiding encryption schemes. In *Usenix Security*, 2014.

[18] Daniel Luchaup, Thomas Shrimpton, Thomas Ristenpart, and Somesh Jha. Formatted encryption beyond regular languages. In *ACM CCS*, 2014.

[19] Dag Arne Osvik, Adi Shamir, and Eran Tromer. Cache attacks and countermeasures: the case of AES. ePrint Report 2005/271, 2005. `http://eprint.iacr.org/2005/271`.

[20] Thomas Ristenpart and Scott Yilek. The Mix-and-Cut Shuffle: Small-domain Encryption Secure for N Queries. In *CRYPTO*, 2013.

[21] Matthew Skala. Hypergeometric tail inequalities: ending the insanity. `http://arxiv.org/pdf/1311.5939v1.pdf`.

[22] HP Security Voltage. HP format-preserving encryption. `https://www.voltage.com/technology/data-encryption/hp-format-preserving-encryption/`.

[23] Wikipedia. Tokenization. `https://en.wikipedia.org/wiki/Tokenization_(data_security)`.

A Proof of Theorem 3

We now prove the security of Zig-Zag as a SPRP for the domain completion setting as stated in Theorem 3. The complexity-theoretic steps of the proof (i.e., constructing the reductions $\mathcal{B}$ and $\mathcal{C}$) are straightforward, so we will omit them and only prove the information-theoretic step.

We start with a standard lemma.

Lemma 4 *Let A be any set and B a subset. Then*

$$x \leftarrow\!\!\$\ A \setminus B \approx x \leftarrow\!\!\$\ A, \text{if } x \in B \text{ then } x \leftarrow\!\!\$\ A \setminus B$$

Informally, the distribution of x sampled from $A \setminus B$ is the same as sampling x from A directly, then re-sampling if $x \in B$.

Proof: Let $|A| = a$ and $|B| = b$. The probability that the output x' takes any particular value c on the left-hand side is $\frac{1}{a-b}$. Using the law of total probability we can partition the probability space of the right-hand side into events conditioned on either $x \in B$ or its complement. Formally, it is $\Pr[x' = c | x \in B] \cdot \Pr[x \in B] + \Pr[x' = c | x \notin B] \cdot \Pr[x \notin B]$. This is equal to $\frac{1}{a-b} \cdot \frac{b}{a} + \frac{1}{a-b} \cdot (1 - \frac{b}{a})$. Multiplying and cancelling gives you $\frac{1}{a-b}$, so they are the same.

Now, we can state and prove the main lemma, which says that no adversary has any advantage in distinguishing a random permutation from the Zig-Zag construction instantiated with two random permutations in place of F and $\overline{E}$.

Lemma 5 *Let $\mathcal{A}$ be any adversary and Π_1, Π_2 be two permutations sampled uniformly at random from* $\mathrm{Perm}(\mathcal{M})$*, the set of all permutations on $\mathcal{M}$. Then*

$$\mathbf{Adv}^{\mathrm{sprp}}_{E^{zz}_{\mathcal{T},\Pi_1,\Pi_2}}(\mathcal{A}) = 0\ .$$

Proof: The starting game is game G0 of Figure 14 and the ending game is game G4 of that figure. In each game, a permutation over all of $\mathcal{M}$ is constructed. We have omitted the explicit construction of the Enc and Dec oracles in some games; these just use the forward and reverse table lookups for the permutation returned from FullPerm or GetPerm. The transition that justifies sampling the entire permutation at once (instead of lazy-sampling) is omitted, but this is standard.

Game G0 → Game G1 The only code change from G0 to G1 is the boxed code. The boxed statement in G1 is equivalent to the check $\Pi_1(c_1) \in \mathcal{T}$ because the code is just checking c_1 against the images of $\mathcal{T}$ instead of decrypting it and checking if the preimage is in $\mathcal{T}$.

Game G1 → Game G2 From Lemmas 2 and 3 it is clear that once we call Π_2 on a point, we can effectively exclude it from consideration for future samples. Thus, the transition from G1 to G2 is conservative.

Game G2 → Game G3 By Lemma 4, the distribution of samples of the value c_1 in the second 'for' loop of games G2 and G3 are the same. Because in G3 we're excluding all possible values that could collide with previously-sampled points in the images of Π_2 and Π_1, the 'while' loop is never called, so we can remove it in G3.

Game G3 → Game G4 This is a straightforward transition — simply combine taken_1 and taken_2 into one set of points that is defined at the beginning of the procedure. This does not change the distribution of points because we choose the image set of $\mathcal{T}$ first, so all points in taken_1 are fixed at the beginning of the second loop of FullPerm. There is another intermediate transition to game G4 of Figure 14 that is not depicted — moving from choosing the points of $\mathcal{T}$ first to choosing all points of $\mathcal{M}$ in order. However, it is not hard to see that the distribution on π is the same in either case.

The adversary's distinguishing advantage in all transitions is zero, since every transition is conservative. Thus, the lemma follows. ∎

B Proof of Theorem 10

Proof: We need two computational steps to transition from game Real_E of Figure 10 to game Sim_E of Figure 15, so the distinguishing advantage is upper-bounded by $\mathbf{Adv}^{\mathrm{sprp}}_{\overline{E}}(\mathcal{B}) + \mathbf{Adv}^{\mathrm{sprp}}_{F}(\mathcal{C})$. We must prove that

$$\Pr\left[\,\mathrm{Ideal}^{\mathcal{A}} \Rightarrow 1\,\right] = \Pr\left[\,\mathrm{Sim}^{\mathcal{A}} \Rightarrow 1\,\right]$$

To do this, we will reproduce part of Lemma 5 above. We will use a sequence of game transitions starting from $\mathrm{Sim}^{\mathcal{A}}$ and ending at $\mathrm{Ideal}^{\mathcal{A}}$. Our intermediate transitions will be to games G1, G2, and G3 in Figure 15. We will justify each transition in sequence.

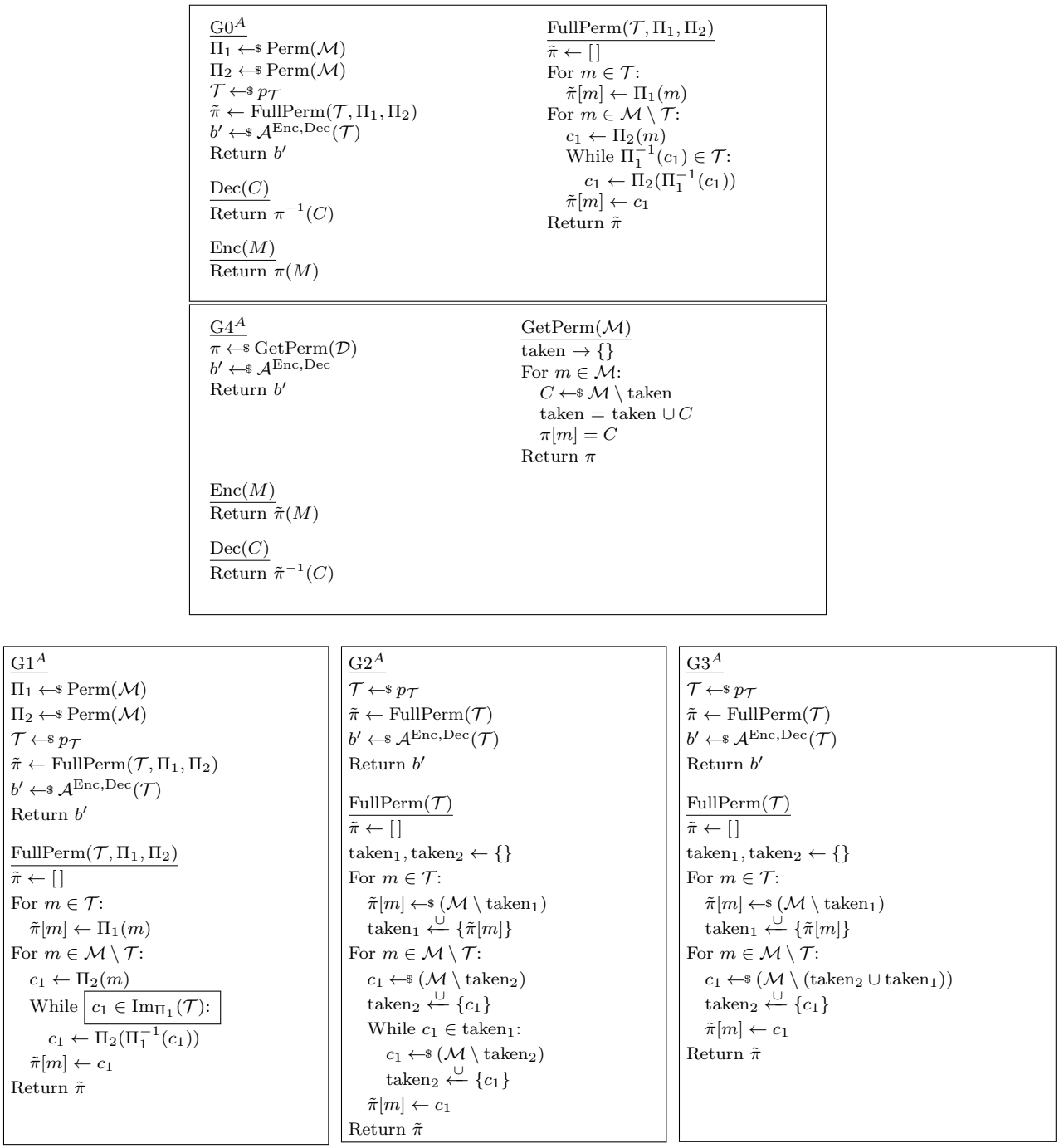

Figure 14: Games for Lemma 5. The boxed code in game G1 is not excluded from any game.

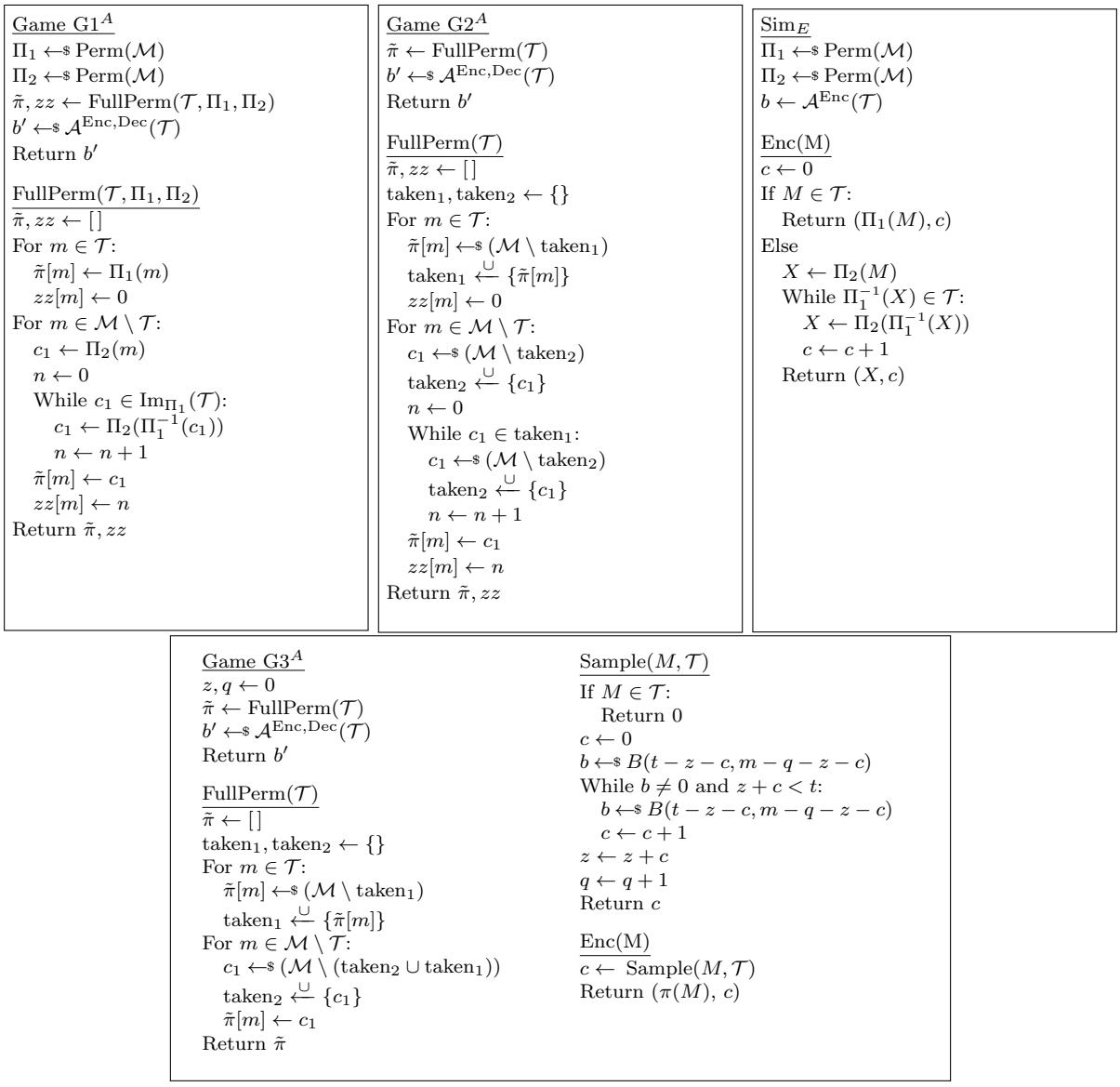

Figure 15: Games for the proof of Theorem 10. The GetPerm procedure is as in Figure 14 above. In all games, $t = |\mathcal{T}|$. Unless otherwise specified, the Enc and Dec oracles queries are answered in the obvious way using $\tilde{\pi}$.

$\mathbf{Sim}^{\mathcal{A}} \rightarrow$ Game G1 The only change in the game code is the condition in the while loop in FullPerm. We argued in Lemma 5 that this does not change the distribution of permutations output by FullPerm.

Game G1 $\rightarrow$ Game G2 Follows by the same proof in Lemma 5 above. By that earlier result, the distribution over the permutations is the same in both games. Since the value n is sampled along with each point of the permutation, it also has the same distribution in both games.

Game G2 $\rightarrow$ Game G3 This step is crucial because in this transition we are no longer performing any zig-zags, so we must prove the distribution over the return values of the Sample procedure in game G3 is identical to those stored in the zz array in game G2. Let's start by looking at the probability of the first c_1 drawn has $c_1 \in \text{taken}_1$ for any point in game G2's FullPerm procedure. If there have been z previous zig-zags, there are $|\text{taken}_1| - z$ points we could still sample to get $c_1 \in \text{taken}_1$. But there are $m - |\text{taken}_2| - z$ total points we could select, so the probability of the first $c_1 \in \text{taken}_1$ is $\frac{|\text{taken}_1|-z}{m-|\text{taken}_2|-z}$. The probability of the next c_1 also being in taken_1 is $\frac{|\text{taken}_1|-z-1}{m-|\text{taken}_2|-z-1}$. In general, the probability of the nth c_1 sampled being in taken_1 is $\frac{|\text{taken}_1|-z-c}{m-|\text{taken}_2|-z-c}$. But this is exactly the distribution we're drawing from in game G3.

Game G3 $\rightarrow$ $\mathbf{Ideal}^{\mathcal{A}}$ The Sample procedure is the same in both games, and the GetPerm procedure is as in Figure 14. Our proof of that theorem applies here as well, so the distribution over the permutations and the outputs of the Sample procedure is the same in both games. This is therefore a conservative game transition, since the distributions are identical.

Since each game transition is conservative, the result follows.

C Proof of Theorem 5

We start the proof of Theorem 5 with the core information-theoretic argument, captured by the next lemma. In the lemma, the notation $E^{zz}[\pi, \overline{\pi}]$ means a Zig-Zag domain-extended cipher (in the sense of Section 6) implemented with random permutations π and $\overline{\pi}$ in the place of F and $\overline{E}$, respectively.

Lemma 6 *Let $\mathcal{A}$ be an SEPRP adversary against $E^{zz}[\pi, \overline{\pi}]$.*

$$\mathbf{Adv}^{seprp}_{E^{zz}[\pi,\overline{\pi}]}(\mathcal{A}) = 0$$

Proof: We claim that Zig-Zag implemented with two random permutations is identical to our ideal object. We've written the process of choosing the entire permutation at once for each game. In game 0, rather then iterating over every point in $\mathcal{M}$ in order, we choose the points in $\mathcal{T}$ first. This is a 'conservative' change in the sense of [4] because the only random choices in the algorithm are $\Pi_{\mathcal{D}}$ and $\Pi_{\mathcal{M}}$. Because they are fixed when the function is called, we can fill in the points of the permutation in any order we choose. We will justify each of the intermediate game transitions in sequence. As in Lemma 5, after the permutation on $\mathcal{M}$ is sampled we use forward and reverse table lookups for encryption and decryption, respectively. Thus, the proof implies indistinguishability of encryption and decryption. We omit the game transition that justifies sampling a permutation over the entire domain — since our adversaries are information-theoretic they can compute FullPerm and ExtPerm.

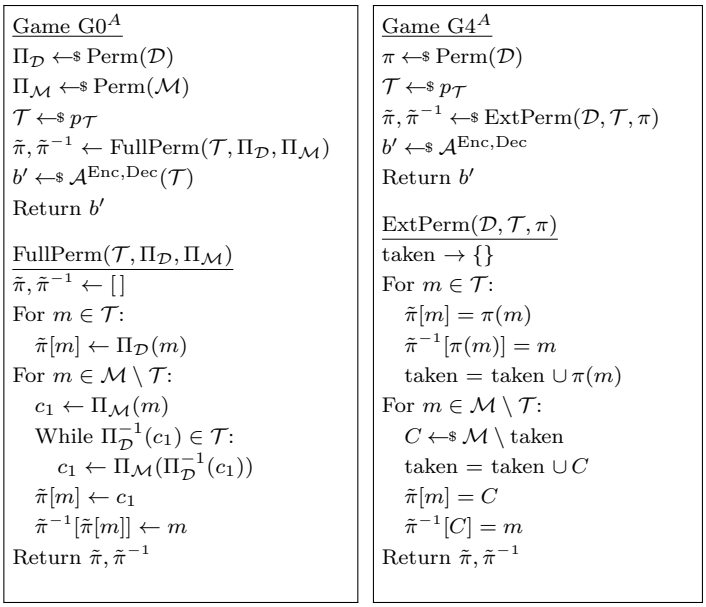

Figure 16: Games for Lemma 6. Enc and Dec queries are answered using $\tilde{\pi}$. Intermediate games G1, G2, and G3 are in Figure 17.

Game G0 → Game G1 The only code change from G0 to G1 is the boxed code. The boxed statement in G1 is equivalent to the check $\Pi_{\mathcal{D}}(c_1) \in \mathcal{T}$ because you're just checking c_1 against the images of $\mathcal{T}$ instead of decrypting it and checking if the preimage is in $\mathcal{T}$.

Game G1 → Game G2 From Lemmas 2 and 3 it is clear that once we call $\Pi_{\mathcal{M}}$ on a point, we can effectively exclude it from consideration for future samples. Thus, the transition from G1 to G2 is conservative.

Game G2 → Game 0c By the Resampling Lemma 4, the distribution of samples of the value c_1 in the second 'for' loop of games G2 and G3 are the same. Because in G3 we're excluding all possible values that could collide with previously-sampled points in the images of $\Pi_{\mathcal{M}}$ and $\Pi_{\mathcal{D}}$, the 'while' loop is never called, so we can remove it in G3.

Game G3 → Game G4 This is a straightforward transition - simply combine $\text{taken}_{\mathcal{D}}$ and $\text{taken}_{\mathcal{M}}$ into one set of points that is defined at the beginning of the procedure. This does not change the distribution of points because we choose the image set of $\mathcal{T}$ first, so all points in $\text{taken}_{\mathcal{D}}$ are fixed at the beginning of the second loop of FullPerm. There is another intermediate transition that is not depicted - we should change game G4 to lazy-sample the points of its original permutation π instead of selecting it during initialization, but since it is a uniformly random permutation of $\mathcal{D}$ in either case it is the same.

The adversary's distinguishing advantage in all transitions is zero, since every transition is conservative. Thus, the lemma follows. ∎

The proof of Theorem 5 uses two game transitions to replace F and $\overline{E}$ with random permutations, then applies Lemma 6. The first two game transitions are standard, and we will omit them. Applying the lemma above completes the proof.

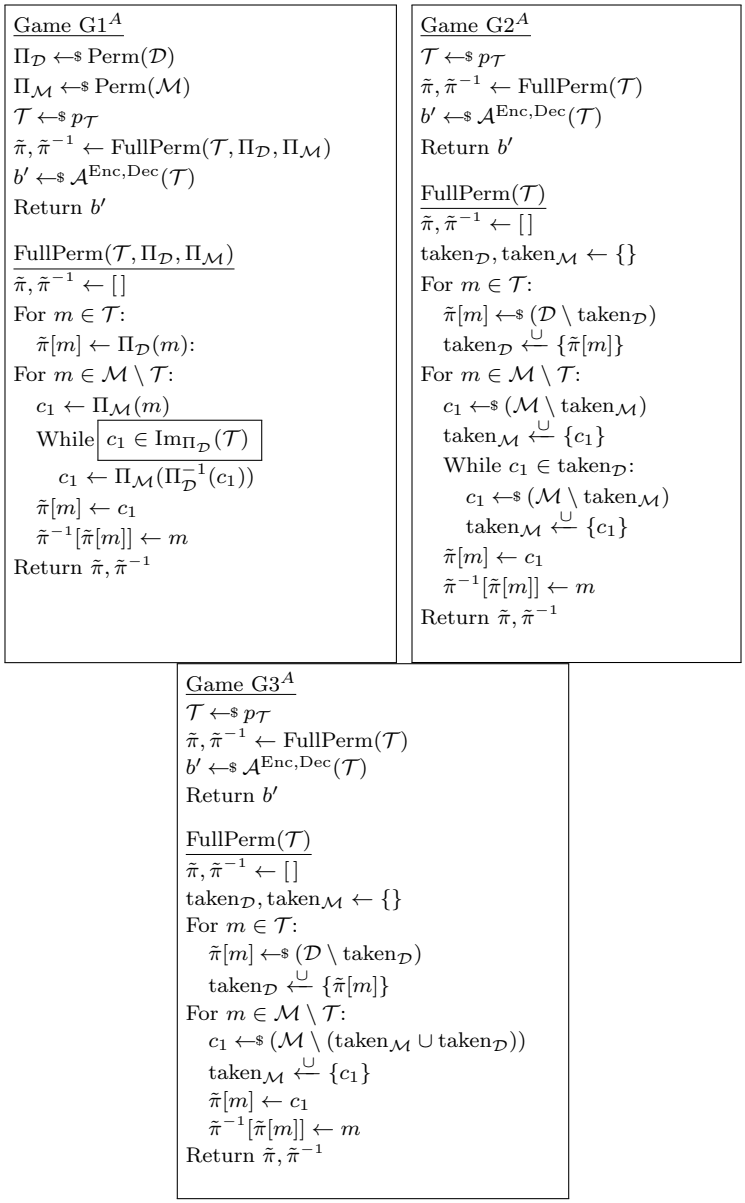

Figure 17: Intermediate games for Lemma 6. Enc and Dec oracle queries are answered using $\tilde{\pi}$. The boxed code in game G1 is not excluded from any game.

D Proof of Theorem 12

To analyze the construction, we will assume tables for the adjusted points have not been created, for ease of exposition. To begin, let $\overline{E}\colon \mathcal{M} \to \mathcal{M}$ be a uniformly random permutation. For a preserved domain set $\mathcal{T} = \{\tau_1, \ldots, \tau_t\}$ and a uniformly random permutation $F\colon \mathcal{D} \to \mathcal{D}$ we construct a sequence of permutations $E^{rzz}_{K^{\text{rzz}}_0}, \ldots, E^{rzz}_{K^{\text{rzz}}_t}$, such that $E^{rzz}_{K^{\text{rzz}}_0} = \overline{E}, E^{rzz}_{K^{\text{rzz}}_i}(\tau_j) = F(\tau_j)$ for all $1 \le j \le i \le t$. Each $E^{rzz}_{K^{\text{rzz}}_i}$ corresponds to the lookup table Emap for E^{rzz} after the ith iteration of KT^{rzz}. Note that we will abuse the notation slightly below, since if $\overline{E}$ and F are random permutations there will be no keys generated in KT^{rzz}; we refer to the straightforward modification of KT^{rzz} with random permutations. We will take each step of the transformation at a time. For each step, as above, we need to look at three different cases:

Case 1: If $E^{rzz}_{K^{\text{rzz}}_{i-1}}(\tau_i) = F(\tau_i)$, we set $E^{rzz}_{K^{\text{rzz}}_i} = E^{rzz}_{K^{\text{rzz}}_{i-1}}$.

Case 2: If $E^{rzz}_{K^{\text{rzz}}_{i-1}}(\tau_i) \in \mathcal{D} \setminus \{F(\tau_i)\}$ we set:

$$E^{rzz}_{K^{\text{rzz}}_i}(x) = \begin{cases} F(\tau_i) & x = \tau_i \\ E^{rzz}_{K^{\text{rzz}}_{i-1}}(\tau_i) & x = D^{rzz}_{K^{\text{rzz}}_{i-1}}(F(\tau_i)) \\ E^{rzz}_{K^{\text{rzz}}_{i-1}}(x) & \text{otherwise} \end{cases}$$

Case 3: If $E^{rzz}_{K^{\text{rzz}}_{i-1}}(\tau_i) \in \mathcal{N}$ we pick a *random* $p_i \in \mathcal{D}$ such that $p_i \notin \{\tau_1, \ldots, \tau_{i-1}\}$ and $E^{rzz}_{K^{\text{rzz}}_{i-1}}(p_i) \in \mathcal{D}$. We then set:

$$E^{rzz}_{K^{\text{rzz}}_i}(x) = \begin{cases} F(\tau_i) & x = \tau_i \\ E^{rzz}_{K^{\text{rzz}}_{i-1}}(\tau_i) & x = p_i \\ E^{rzz}_{K^{\text{rzz}}_{i-1}}(p_i) & x = D^{rzz}_{K^{\text{rzz}}_{i-1}}(F(\tau_i)) \text{ and } x \neq p_i \\ E^{rzz}_{K^{\text{rzz}}_{i-1}}(x) & \text{otherwise} \end{cases}$$

Assume that $t \le d - n$, $E^{rzz}_{K^{\text{rzz}}_{i-1}}$ and F are random permutation of the respective domains and $\mathcal{T}$ is a random subset of size t of $\mathcal{D}$. Then, $E^{rzz}_{K^{\text{rzz}}_i}$ is indistinguishable from a random permutation.

To begin, we're given a permutation $\Pi\colon \mathcal{M} \to \mathcal{M}$. We want to show that $\Pr\left[E^{rzz}_{K^{\text{rzz}}_i} = \Pi\right] = 1/(m!)$. Let $\mathcal{S}_i$ be the set of elements in $\mathcal{D}$ that are domain-preserved under $E^{rzz}_{K^{\text{rzz}}_i}$ i.e.

$$\mathcal{S}_i = \{x \in \mathcal{D} | E^{rzz}_{K^{\text{rzz}}_i}(x) \in \mathcal{D}\}$$

and let $s = |\mathcal{S}_0|$. The construction of $E^{rzz}_{K^{\text{rzz}}_i}$ guarantees that $|\mathcal{S}_{i-1}| = |\mathcal{S}_i|$ and, therefore, $s = |\mathcal{S}_i|$ for all i. We also note that $s \ge d - n \ge t$.

We now assume $E^{rzz}_{K^{\text{rzz}}_i} = \Pi$ and look at each of the cases of generating $E^{rzz}_{K^{\text{rzz}}_i}$ to find the probability of this event.

In case 1, we had some $E^{rzz}_{K^{\text{rzz}}_{i-1}}$ which happened to be identical to Π and for which we happened to have $E^{rzz}_{K^{\text{rzz}}_{i-1}}(\tau_i) = F(\tau_i)$. By our assumption on $E^{rzz}_{K^{\text{rzz}}_{i-1}}$, the probability of $E^{rzz}_{K^{\text{rzz}}_{i-1}} = \Pi$ is $1/m!$. τ_i cannot be any of $\tau_1, \ldots, \tau_{i-1}$, however because $\mathcal{T}$ is a random subset of $\mathcal{D}$, the probability of τ_i being any other value in $\mathcal{D}$ is $1/(d-i+1)$. Hence, the probability of $\tau_i \in \mathcal{S}_{i-1}$ is $(s-i+1)/(d-i+1)$. (Note that $\tau_1, \ldots, \tau_{i-1}$ are all in $\mathcal{S}_{i-1}$.) Last, for $\tau_i \in \mathcal{S}_{i-1}$, the probability of $E^{rzz}_{K^{\text{rzz}}_{i-1}}(\tau_i) = F(\tau_i)$ is $1/(d-i+1)$. Thus, the probability of generating $E^{rzz}_{K^{\text{rzz}}_i} = \Pi$ using the first case is

$$\frac{s-i+1}{m!(d-i+1)^2}$$

In case 2, we had $\tau_i \in \mathcal{S}_{i-1}$. The probability of that happening is $(s-i+1)/(d-i+1)$. We also had $E^{rzz}_{K^{\text{rzz}}_{i-1}}(\tau_i) \neq F(\tau_i)$. The conditional probability of that happening, given that $\tau_i \in \mathcal{S}_{i-1}$, is $(d-i)/(d-i+1)$. For each choice of $\tau_i \in \mathcal{S}_{i-1}$ and $E^{rzz}_{K^{\text{rzz}}_{i-1}}(\tau_i)$, there is exactly one permutation that applying the second case would result in Π. Hence, the probability of generating $E^{rzz}_{K^{\text{rzz}}_{i}} = \Pi$ using the second case is

$$\frac{(s-i+1)(d-i)}{m!(d-i+1)^2}$$

In case 3 we had $\tau_i \notin \mathcal{S}_{i-1}$. The probability for this is $(d-s)/(d-i+1)$. For each combination of p_i, $E^{rzz}_{K^{\text{rzz}}_{i-1}}(\tau_i)$ and $E^{rzz}_{K^{\text{rzz}}_{i-1}}(p)$, there is exactly one permutation that applying the third case will result in Π. Hence, the probability of generating $E^{rzz}_{K^{\text{rzz}}_{i}} = \Pi$ using the third case is

$$\frac{d-s}{m!(d-i+1)}$$

The probability of $E^{rzz}_{K^{\text{rzz}}_{i}} = \Pi$ is the sum of the probabilities for each separate case because the three cases are mutually exclusive. Hence

$$\Pr\left[E^{rzz}_{K^{\text{rzz}}_{i}} = \Pi\right] = \frac{s-i+1}{m!(d-i+1)^2} + \frac{(s-i+1)(d-i)}{m!(d-i+1)^2} + \frac{d-s}{m!(d-i+1)} = \frac{1}{m!}$$

We've showed that each $E^{rzz}_{K^{\text{rzz}}_{i}}$ is indistinguishable from a random permutation. Thus, the remainder of the proof follows from induction on i. ∎