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 [4, 6, 11, 16, 17, 20] as well as widespread use in industry [12, 18, 23, 25].

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 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 $F_{K_\mathsf{o}} : \mathcal{M} \to \mathcal{M}$ for a "key" K_o 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 [4, 11, 16], 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 $F_{K_\mathsf{o}} : \mathcal{D} \to \mathcal{D}$ 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_o} operating on some domain and, using knowledge of K_o, build a new cipher E_K such that $E_K(M) = F_{K_\mathsf{o}}(M)$ 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

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

Fig. 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.

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 $F_{K_o}(M)$. 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 $E_K(M) = F_{K_o}(M')$.

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_o} 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 [13] is straightforward.

The second case is more nuanced. We first observe that *no* scheme can achieve SPRP security 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 [4], 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 [4]: 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.

Full version. Due to space constraints, we had to omit a number of proofs. These will be available in the full version, which will be available from the authors' websites.

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 [4] is a generalization of ciphers that allows multiple lengths as well as tweaks [13]. 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 $\{f(x) \mid x \in \mathcal{X}\}$.

SPRP security. Ciphers are considered secure if they behave like strong pseudorandom permutations (SPRPs). Let 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. 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.

```
main SPRP1$_E^{\mathcal{A}}$
  $K \leftarrow\$ \mathcal{K}$
  $b' \leftarrow \mathcal{A}^{\mathrm{Enc,Dec}}$
  return $b'$

proc Enc($M$)
  return $E_K(M)$

proc Dec($C$)
  return $E_K^{-1}(C)$
```

```
main SPRP0$_E^{\mathcal{A}}$
  $\pi \leftarrow\$ \mathrm{Perm}(\mathcal{D})$
  $b' \leftarrow \mathcal{A}^{\mathrm{Enc,Dec}}$
  return $b'$

proc Enc($M$)
  return $\pi(M)$

proc Dec($C$)
  return $\pi^{-1}(C)$
```

Fig. 2. SPRP security games for a cipher E.

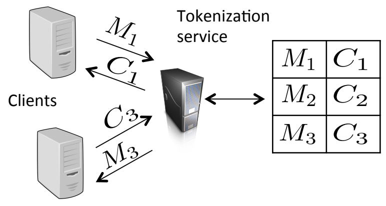

Fig. 3. Tokenization system after choosing random values C_1, C_2, C_3 for plaintexts $\mathcal{T} = \{M_1, M_2, M_3\}$.

We assume that adversaries do not repeat any
oracle queries and do not ask queries to which they already knows the answer, like querying a decryption oracle with the result of a previous encryption oracle query.

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. We formalize this as the following lemma. A full proof of this lemma can be found in [8, 22] but is omitted here.

Lemma 1. *Let X be a hypergeometrically distributed random variable and n be the number of samples. Then for any real number t, $\Pr[E[X]+tn \leq X] \leq e^{-2t^2 n}$, 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 [27] 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 like 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 $F_{K_o}(M)$. Here K_o is just an explicitly stored table of message, token pairs. The token $F_{K_o}(M)$ 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. Primarily 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.) 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 $E_K(M) = F_{K_o}(M)$ 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 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 $KT(K_o, \mathcal{T})$ and returning the result.

A domain-completed cipher DCC preserves a point M if $E_K(M) = F_{K_o}(M)$ 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 or that security should hold even if the adversary knows $\mathcal{T}$.

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 [4]. With permission we reproduce it here. Recall that a rank-encipher-unrank construction uses a ranking function $\mathit{rank}\colon\ \mathcal{M} \to \mathbb{Z}_m$, which is a bijection with inverse $\mathit{unrank}\colon\ \mathbb{Z}_m \to \mathcal{M}$. 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 [4].) 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_o} on $\mathcal{T}$. The construction uses a cipher $\overline{E}\colon\ \mathbb{Z}_d \to \mathbb{Z}_d$ and a ranking function $\mathit{rank}\colon\ \mathcal{M} \to \mathbb{Z}_m$ with inverse unrank. The construction builds from rank rankings $\mathit{rank}_\mathcal{D}\colon\ \mathcal{D} \to \mathbb{Z}_d$ and $\mathit{rank}_\mathcal{R}\colon\ \mathcal{R} \to \mathbb{Z}_d$. 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 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 $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_t) \in \mathbb{Z}_m^t$. Similarly it computes $\{\mathit{rank}(Y) \mid Y \in \mathsf{Img}_{F_{K_\mathsf{o}}}(\mathcal{T})\}$ and sorts it to obtain a list $\boldsymbol{y} = (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_t) \in \mathbb{Z}_m^t$. It also chooses a new key $\overline{K}$ for a helper cipher $\overline{E}$ on $\mathbb{Z}_d$ and outputs $K = (K_\mathsf{o}, \overline{K}, \boldsymbol{x}, \boldsymbol{y})$.

Enciphering is performed via $E_K^{rte}(X) = \mathit{unrank}_\mathcal{R}(\overline{E}_{\overline{K}}(\mathit{rank}_\mathcal{D}(X)))$ 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 $\boldsymbol{x}_i < x$, 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 $\boldsymbol{x}_j - j + 1 \le x'$, and then outputting $X \leftarrow \mathit{unrank}(x' + j)$. The construction of $\mathit{rank}_\mathcal{R}$ is similar, using $\boldsymbol{y}$ instead of $\boldsymbol{x}$.

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 [4], efficient rankings can be built for most formats of practical interest [4,11,16,17]. The additional overhead of removing the $\mathcal{T}$ (resp. $\mathsf{Img}_{F_{K_\mathsf{o}}}(\mathcal{T})$) 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 affords various flexibilities 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, $\overline{E}$).

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 be in some cases tricky to implement without side-channels, if one uses the table-based constructions for ranking regular languages [4, 11, 16] or context-free grammars [17].

5 The Zig-Zag Construction for Domain Completion

In this section we will introduce an algorithm that achieves SPRP security in the domain completion setting. 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 $(\mathcal{T}, K_{\mathsf{o}}, \overline{K})$ 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 $(\mathcal{T}, K_{\mathsf{o}}, \overline{K})$ define a key for the enciphering algorithm E^{zz} and deciphering algorithm D^{zz}, detailed in Figure 4.

```
$E^{zz}_{\mathcal{T},K_{\mathsf{o}},\overline{K}}(M)$:
if ($M \in \mathcal{T}$ ):
    return $F_{K_{\mathsf{o}}}(M)$
else
    $i \leftarrow 1$
    $M_0 \leftarrow M$
    while ( $M_{i-1} \in \mathcal{T}$ )
        $Y_i \leftarrow \overline{E}_{\overline{K}}(M_{i-1})$
        $M_i \leftarrow F^{-1}_{K_{\mathsf{o}}}(Y_i)$
        $i \leftarrow i+1$
    return $Y_{i-1}$
```

```
$D^{zz}_{\mathcal{T},K_{\mathsf{o}},\overline{K}}(C)$:
if ( $F^{-1}_{K_{\mathsf{o}}}(C) \in \mathcal{T}$ ):
    return $F^{-1}_{K_{\mathsf{o}}}(C)$
else
    $i \leftarrow 1$
    $M_i \leftarrow \overline{D}_{\overline{K}}(C)$
    while ( $M_i \in \mathcal{T}$ or $i = 1$ ):
        $Y_{i+1} \leftarrow F_{K_{\mathsf{o}}}(M_i)$
        $M_{i+1} \leftarrow \overline{D}_{\overline{K}}(Y_{i+1})$
        $i \leftarrow i+1$
    return $M_i$
```

Fig. 4. Zig-Zag encryption and decryption algorithms.

Towards building intuition about the Zig-Zag construction, 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 [6]. 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 $Y = \overline{E}_{\overline{K}}(X) \in \mathsf{Img}_{F_{K_{\mathsf{o}}}}(\mathcal{T})$. 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 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., $\overline{E}_{\overline{K}}(Y)$.

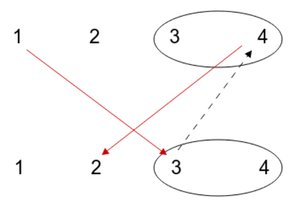

Fig. 5. Graphical depiction of a zig-zag. The domain (first row) has a target set $\mathcal{T} = \{3, 4\}$. We encrypt 1 to 3, which collides with the ciphertext in the image of $\mathcal{T}$. We then decrypt to get $4 \in \mathcal{T}$ and re-encrypt 4 to get 2.

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 the diagram of 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 $F^{-1}_{K_\circ}$. We start by calling $\overline{E}_{\overline{K}}(1)$, which (say) gives us the image of a point in $\mathcal{T}$. We call F^{-1} and find that the preimage of this point is 4. We then call $\overline{E}_{\overline{K}}(4)$, which gives us $E^{zz}_{\overline{K}}(1) = \overline{E}_{\overline{K}}(4) = 2$.

5.1 Running time of the Zig-Zag construction

The Zig-Zag construction, in the worst-case, requires $|\mathcal{T}|$ iterations of the while loop. First, we note that if the algorithm enters the while loop, it must exit — otherwise permutivity of $\overline{E}_{\overline{K}}$ would be violated. In the worst case, then, the while loop will hit every point in $\mathcal{T}$. In the full version we provide a formal proof of this. We also show that, when encrypting the entire domain $\mathcal{M}$ under $E^{zz}_{\overline{K}}$, $\overline{E}$ is executed at most once per point in $\mathcal{M}$. 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).

This doesn't mean that for individual points the running time is not significantly delayed (in the worst case, requiring $2{\cdot}|\mathcal{T}|$ underlying cipher calls). But in fact the expected running time for an arbitrary point is small, as captured by the next theorem, and assuming the underlying ciphers are random permutations.

Theorem 1. *Let E^{zz} be implemented as in Figure 4 except with F replaced with Π_1 and $\overline{E}$ replaced with Π_2, where Π_1 and Π_2 are random permutations over $\mathcal{M}$. Let I be a random variable denoting the number of iterations of the inner 'while' loop of E^{zz} with domain $\mathcal{M}$ and preservation set $\mathcal{T}$ taken when enciphering an arbitrary point in $\mathcal{M} \setminus \mathcal{T}$. Let $|\mathcal{T}| = t$. Then, if $t \leq |\mathcal{M}|/2$, it holds that* $\mathrm{E}\,[\,I\,] \leq 2$.

Proof. Consider an arbitrary $M \in \mathcal{M}$. First if $M \in \mathcal{T}$ then the number of iterations of the while loop is zero. Consider otherwise, and let the transcript of points defined in the while loop be $\mathcal{P} = \{M_0, Y_1, M_1, Y_2, M_2, \ldots, Y_i, M_i\}$. Then the size of this transcript is a random variable, over the coins used to choose Π_1, Π_2, and we denote it by I. We have that $\Pr[I > t] = 0$ by our arguments of correctness discussed above. Then for any $1 \le j \le t$, because Π_1, Π_2 are random permutations independent of M, we have that

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

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

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

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

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

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 $2 \le j \le t$, 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 $\mathrm{E}\left[\, I \,\right] \le 2$ for $t \le \frac{m}{2}$. ▮

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

Security of the Zig-Zag construction. In the domain completion setting, our Zig-Zag construction achieves SPRP security. We prove the following theorem in the full version. The proof proceeds via standard reductions to move to an information-theoretic setting in which F and $\overline{E}$ are replaced by random permutations. Then one performs an analysis to show that the Zig-Zag domain-completed cipher, when using random permutations, exactly defines a random permutation.

Theorem 2. *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*

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

Adversaries $\mathcal{B}$ and $\mathcal{C}$ each run in time that of $\mathcal{A}$ plus negligible 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 might also be able to 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. Ideally 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 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, 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. We can measure security for a domain-completed cipher $\mathrm{DEC} = (KT, E)$ via the SPRP advantage of 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 $\mathcal{T}$.

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

Theorem 3. *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 a fast, specific SPRP adversary $\mathcal{A}$ that makes $q \leq t$ queries for which*

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

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 an domain extension setting $(F, \mathcal{D}, \mathcal{T}, \mathcal{M})$, an adversary $\mathcal{A}$ and domain-extended cipher DEC $= (KT, E)$, the $\text{SEPRP0}^{\mathcal{A}}_{\text{DEC}}$ and $\text{SEPRP1}^{\mathcal{A}}_{\text{DEC}}$ games in Figure 6 capture what we call "indistinguishability from an extended random permutation". (The games are implicitly parameterized by the domain extension setting.) There $\text{ExtPerm}(\mathcal{D}, \mathcal{T}, \pi)$ 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

```
main SEPRP1$^{\mathcal{A}}_{\text{DEC},\mathcal{T}}$
  $K_o \leftarrow\$ \mathcal{K}$
  $K \leftarrow\$ KT(K_o, \mathcal{T})$
  $b' \leftarrow \mathcal{A}^{\text{Enc},\text{Dec}}(\mathcal{T})$
  return $b'$

proc Enc($M$)
  return $E_K(M)$

proc Dec($C$)
  return $E_K^{-1}(C)$
```

```
main SEPRP0$^{\mathcal{A}}_{\text{DEC},\mathcal{T}}$
  $\pi \leftarrow\$ \text{Perm}(\mathcal{D})$
  $\tilde{\pi} \leftarrow\$ \text{ExtPerm}(\mathcal{D}, \mathcal{T}, \pi)$
  $b' \leftarrow \mathcal{A}^{\text{Enc},\text{Dec}}(\mathcal{T})$
  return $b'$

proc Enc($M$)
  return $\tilde{\pi}(M)$

proc Dec($C$)
  return $\tilde{\pi}^{-1}(C)$
```

Fig. 6. Games defining SEPRP security.

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

Zig-Zag for domain extension. We now consider the security of the Zig-Zag construction in the domain extension setting. Fixing a setting $(F, \mathcal{D}, \mathcal{T}, \mathcal{M})$, observe that the construction ZZ $= (KT^{zz}, E^{zz})$ from Section 5 provides a domain-extended cipher. The next theorem captures its SEPRP security. The proof appears in the full version of the paper.

Theorem 4. *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*

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

The adversaries $\mathcal{B}, \mathcal{C}$ *each run in time at most that of* $\mathcal{A}$ *plus a negligible 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. In particular we'll explore the relationship between SEPRP and the notions of message recovery and message privacy security for ciphers introduced by Bellare et al. [4]. 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 [4] 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 it 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 $\mathbf{Eq}$ 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

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

where the rightmost term is defined over $\mathbb{S}_{q'}$, the set of all simulators making at most q' queries to their $\mathbf{Eq}$ oracles. In what follows, the simulator can depend

```
main gMR_DEC
    $K_o \leftarrow\$ \mathcal{K}$
    $K \leftarrow\$ KT(K_o, \mathcal{T})$
    $X^* \leftarrow\$ \mathcal{A}_1(\mathcal{T})$
    $Y^* \leftarrow E_K(X^*)$
    $X \leftarrow \mathcal{A}_2^{\mathbf{Enc}}(\mathcal{T}, Y^*)$
    return $(X = X^*)$

Enc(X)
    return $E_K(X)$
```

```
main gMRI
    $X^* \leftarrow\$ \mathcal{A}_1(\mathcal{T})$
    $X \leftarrow \mathcal{S}^{\mathbf{Eq}}(\mathcal{T}, \mathrm{aux}(X^*))$
    return $(X = X^*)$

Eq(X)
    return $(X = X^*)$
```

```
main gMP_DEC
    $K_o \leftarrow\$ \mathcal{K}$
    $K \leftarrow\$ KT(K_o, \mathcal{T})$
    $X^* \leftarrow\$ \mathcal{A}_1(\mathcal{T})$
    $Y^* \leftarrow E_K(X^*)$
    $Z \leftarrow \mathcal{A}_2^{\mathbf{Enc}}(\mathcal{T}, Y^*)$
    return $(Z = \mathcal{A}_3(X^*))$

Enc(X)
    return $E_K(X)$
```

```
main gMPI
    $X^* \leftarrow\$ \mathcal{A}_1(\mathcal{T})$
    $X \leftarrow \mathcal{S}^{\mathbf{Eq}}(\mathcal{T}, \mathrm{aux}(X^*))$
    return $(X = \mathcal{A}_3(X^*))$

Eq(X)
    return $(X = X^*)$
```

Fig. 7. Generalized Message-recovery and message privacy games.

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 like 1 or 2 times q. [3] 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. All reductions below are tight up to some small constant factors.

Since MR security is shown not to imply SPRP security in [4], we expect that it does not imply SEPRP security. 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 $d+1$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, $\mathbf{Adv}_E^{\mathrm{gmr}}(\mathcal{A}, q, \mathrm{B}) \geq \frac{1}{2}$, where 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 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

[3] Note that when $q' = q$ and aux is the function that always outputs $\perp$, this definition corresponds exactly to the message recovery definition from [4].

when the simulator is given some auxiliary information about the hidden point, namely whether or not it is in $\mathcal{T}$.

Theorem 5. *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 such that*

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

where $\mathrm{aux}(X)$ *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 $\mathcal{A}_2(\mathcal{T}, \mathrm{Enc}(X^*))$, simulating $\mathcal{A}_2$'s **Enc** oracle using its own encryption oracle for the SEPRP game. When $\mathcal{A}_2$ outputs its guess X, $\mathcal{B}$ returns 1 if $(X = X^*)$ and 0 otherwise. By construction

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

To complete the proof we must show that

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

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 roughly equivalent to speeding up a guessing attack by a factor of two.

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 *queries,*

$$\mathbf{Adv}^{\mathrm{gmr}}_{\mathrm{DEC}}(\mathcal{A}, 2(q+1), \mathrm{B}) \leq \mathbf{Adv}^{\mathrm{gmr}}_{\mathrm{DEC}}(\mathcal{A}, q, \mathrm{aux})$$

where aux *and* 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

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

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 [4] 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 $\mathcal{A}$ as

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

We will use this generalized definition instead of the one used in [4] 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 [4], which corresponds to our generalized definition when $q = q'$ and $\text{aux} = \perp$.

Theorem 7. *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*

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

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

$$\mathcal{A}_3(m) = \begin{cases} 1 & \text{The message is in } \mathcal{D} \\ 0 & \text{The message is in } \mathcal{M} \setminus \mathcal{D} \end{cases}$$

$\mathcal{A}$ 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 $\max(\frac{d}{m}, \frac{n}{m})$. ∎

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 8. *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 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. ∎

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 (e.g. the Zig-Zag construction) is whether this gives rise to any kind of side-channel attack. Timing side-channels have proven particularly dangerous in applications [5,7] so we would like to prove the Zig-Zag construction 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. ≪**PaulG 8.1:** If we get a chance to make some changes to camera-ready, fix the references to domain extension below. It should be domain completion.≫ Fix some domain extension setting $(F, \mathcal{D}, \mathcal{T}, \mathcal{M})$

```
main Real_ZZ
$K_o \leftarrow\$ \mathcal{K}$
$(\mathcal{T}, K_o, \overline{K}) \leftarrow\$ KT(K_o, \mathcal{T})$
$b \leftarrow \mathcal{A}^{\mathrm{Enc}}$

proc Enc(M)
$c \leftarrow 0$
If $M \in \mathcal{T}$ then
    return $(F_{K_o}(M), c)$
else
    $X \leftarrow \overline{E}_{\overline{K}}(M)$
    while $F_{K_o}^{-1}(X) \in \mathcal{T}$:
        $X \leftarrow \overline{E}_{\overline{K}}(F_{K_o}^{-1}(X))$
        $c \leftarrow c + 1$
return $(X, c)$
```

```
Main Ideal
$z \leftarrow 0$
$q \leftarrow 0$
$\pi \leftarrow \mathrm{GetPerm}(\mathcal{T})$
$b \leftarrow \mathcal{A}^{\mathrm{Enc}}(\mathcal{T})$

proc Sample($M, \mathcal{T}$)
If $(M \in \mathcal{T})$ then return 0
$c \leftarrow 0$
$b \leftarrow\$ B(t - z - c, m - q - z - c)$
while $b \neq 0$:
    $b \leftarrow\$ B(t - z - c, m - q - z - c)$
    $c \leftarrow c + 1$
$z \leftarrow z + c$
$q \leftarrow q + 1$
return $c$

proc Enc(M)
$c \leftarrow\$$ Sample($M, \mathcal{T}$)
return $(\pi(M),\ c)$
```

Fig. 8. Games defining SPRP-with-timing advantage for Zig-Zag.

and let ZZ $= (KT^{zz}, E^{zz})$ be the Zig-Zag construction for it. We define two games, detailed in Figure 8. The first, $\mathrm{Real}_{\mathrm{ZZ}}^{\mathcal{A}}$, 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, $\mathrm{Ideal}^{\mathcal{A}}$ gives the adversary $\mathcal{A}$ access to an oracle that returns random permutation applied to the message as well as a simulated while-loop count that only uses whether $M \in \mathcal{T}$. Define the SPRP-with-timing advantage of an adversary $\mathcal{A}$ against an ZZ as

$$\mathbf{Adv}_{\mathrm{ZZ}}^{\mathrm{sprp+t}}(\mathcal{A}) = \left|\Pr\left[\mathrm{Real}_{\mathrm{ZZ}}^{\mathcal{A}} \Rightarrow 1\right] - \Pr\left[\mathrm{Ideal}^{\mathcal{A}} \Rightarrow 1\right]\right|$$

The interpretation is that efficient adversaries should not be able to distinguish E^{zz} from a random permutation, even with this additional information. In the Sample procedures, the function $B(x, y)$ generates a random bit that is 1 with probability $\frac{x}{y}$.

The following theorem captures Zig-Zag's security in this new model.

Theorem 9. *Fix a domain extension setting $(F, \mathcal{D}, \mathcal{T}, \mathcal{M})$ and let* ZZ $= (KT^{zz}, E^{zz})$ *be the Zig-Zag domain-extended 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*

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

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

We defer the proof to the full version. This theorem lets us say with 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.

8.1 Other sources of side information

Now that we have shown formally that the timing side-channel of the Zig-Zag construction's inner loop does not leak information to an adversary other than whether or not a point is in $\mathcal{T}$, we will discuss more coarse-grained side channels. We first look at remote timing attacks, where the adversary learns how long it takes to perform the encryption or decryption and from that deduces secret information. For convenience, we only discuss leaks in the encryption algorithm. Similar leaks exist in the decryption algorithm.

The main source of secret-dependent timing variations is the zig-zag operation. Each time the algorithm iterates through the while loop it performs two more encryptions. Thus, timing information discloses the number of times the algorithm iterates. Knowing that the algorithm iterates through the while loop is an indication that $M \notin \mathcal{T}$. We prove above that this is the only information about M leaked to an adversary by this timing information.

Other sources of timing variations that may leak secret information include: the different code path taken for $M \in \mathcal{T}$, the test for $M \in \mathcal{T}$ and potential timing variations in the implementations of F and $\overline{E}$.

A common technique for protecting against timing channels is to pad the computation time. The implementation is modified to ensure that the time between the start of the computation and the delivery of the result is fixed [3, 9]. To avoid any side-channel information, this fixed time must be long enough to accommodate any possible length of computation. Askarov et al. [3] suggest an adaptive approach that ensures that only a small amount of information leaks while adapting to the execution time of the computation.

As discussed in Section 5.1, the Zig-Zag construct iterates, on average, less than one time per encryption. The worst-case scenario, however, is that it iterates t times. Padding to the worst-case scenario incurs a significant performance loss. Yet, failure to pad to the worst-case scenario may result in information leaks. To avoid both excessive padding and information leaks, we can pre-compute the value of the new cipher E^{zz} on all the points that require zig-zagging. That is, on the points of the set:

$$\left\{\overline{D}_{\overline{K}}(F_{K_\circ}(M))|M \in \mathcal{T} \wedge \overline{D}_{\overline{K}}(F_{K_\circ}(M) \notin \mathcal{T}\right\}$$

Informally, these are the points not in $\mathcal{T}$ whose encryptions under $\overline{E}$ are in $\mathrm{Im}_{F_{K_\circ}}(\mathcal{T})$.

Storing the precomputed values takes a space linear in t. However, as the Zig-Zag construction needs to store $\mathcal{T}$, the pre-computation only increases the space requirements by a constant factor and, at the same time, guarantees that the computation of E^{zz} requires at most a single application of either F or of

$\overline{E}$. With the pre-computation, padding can provide an efficient countermeasure for remote timing attacks.

Padding is not an efficient countermeasure for local side-channel attacks. Local adversaries can monitor traces that software execution leaves in the cache or in other microarchitectural components [1,2,15,19]. Constant-time implementations that perform no secret-dependent branches or memory accesses can provide protection for ciphers against local side channel attacks [5,19]. However, such implementations need to access every table element when performing a table access. Thus, for the Zig-Zag construction, the check whether $M \in \mathcal{T}$ would require a time linear in t. Rather than using a constant-time implementation of the cipher, implementer can rely on hardware or operating system based measure to provide protection against local side-channel attacks [14,21,24,26].

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

In a previous section we demonstrated that we cannot achieve SPRP security while preserving a subset of the original domain if the adversary knows which subset is preserved. One can naturally ask, then, if there are weaker adversarial settings in which SPRP 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 non-sensitive 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) 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, we are excluding some nonzero

```
$E^{rzz}_{\mathrm{Emap},\overline{K}}(M)$:
if ( $\mathrm{Emap}[M] \neq \perp$ ):
    return $\mathrm{Emap}[M]$
else
    return $\overline{E}_{\overline{K}}(M)$
```

```
$D^{rzz}_{\mathrm{Emap},\overline{K}}(M)$:
if ( $\mathrm{Emap}^{-1}[M] \neq \perp$ ):
    return $\mathrm{Emap}^{-1}[M]$
else
    return $\overline{E}^{-1}_{\overline{K}}(M)$
```

Fig. 9. Recursive Zig-Zag encryption and decryption algorithms

number of 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). 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 KT^{rzz} is constant for each point of $\mathcal{T}$.

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 9 shows the encryption and decryption algorithms. They consult the lookup table Emap for the existence of the value, returning it if found. Otherwise they return the value of the helper cipher $\overline{E}$. The new key $K = \langle \mathrm{Emap}, \overline{K} \rangle$ output by KT^{rzz} contains the lookup table Emap which is pre-calculated by the key transformation algorithm in Figure 10. We use the notation $\mathrm{Emap}[x]$ to refer to the mapping of the element x under Emap. The notation $\mathrm{Emap}^{-1}[y]$ returns the value X such that $\mathrm{Emap}[X] = y$. If Emap does not provide a mapping for x, the value of $\mathrm{Emap}[x]$ is $\perp$. If there is no point mapped to y in Emap, $\mathrm{Emap}^{-1}[y]$ 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 $\tau_{\mathrm{old}} \leftarrow E^{rzz}_{\mathrm{Emap},\overline{K}}(\tau_i)$ where $E^{rzz}_{\mathrm{Emap},\overline{K}}$ is computed as in Fig-

```
$\underline{KT^{rzz}(K_o, \mathcal{T})}$
$\overline{K} \leftarrow\$ \mathcal{K}$
for $i$ from 0 to $t$:
  $\tau_{\text{old}} \leftarrow E^{rzz}_{\text{Emap},\overline{K}}(\tau_i)$
  if ( $\tau_{\text{old}} = F_{K_o}(\tau_i)$) :
    // Case 1: Do nothing
  if ( $\tau_{\text{old}} \in \mathcal{D}$) :
    // Case 2: set mapping for preimage of $F_{K_o}(\tau_i)$)
    $\tau_m \leftarrow D^{rzz}_{\text{Emap},\overline{K}}(F_{K_o}(\tau_i))$
    $\text{Emap}[\tau_m] \leftarrow \tau_{\text{old}}$
  else:
    // Case 3: chose a $p_i$ and swap through it
    $\tau_m \leftarrow D^{rzz}_{\text{Emap},\overline{K}}(F_{K_o}(\tau_i))$
    //Select a random $p_i$
    //that is domain-preserved under $E^{rzz}_{Emap,\overline{K}}$
    do:
      $p_i \leftarrow\$ \mathcal{D} \setminus \{\tau_1, \ldots, \tau_{i-1}\}$
      $p_i^{\text{old}} \leftarrow E^{rzz}_{\text{Emap},\overline{K}}(p_i)$
    while ($p_i^{\text{old}} \in \mathcal{N}$)
    $\text{Emap}[p_i] \leftarrow \tau_{\text{old}}$
    $\text{Emap}[\tau_m] \leftarrow p_i^{\text{old}}$
  endif
  // Always record $\tau_i$
  $\text{Emap}[\tau_i] \leftarrow F_{K_o}(\tau_i)$
endfor
return (Emap, $\overline{K}$)
```

Fig. 10. The Recursive Zig-Zag key transformation. The original cipher is F. The helper cipher is $\overline{E}$.

ure 9. There are then three cases. We will explain each in turn, referring to the case numbers given in Figure 10.

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

Case 2 ($\tau_{\text{old}} \in \mathcal{D}$): This occurs if E^{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 $F_{K_o}(\tau_i)$ because both $F_{K_o}(\tau_i)$ and τ_{old} are in $\mathcal{D}$. In this case we can do a zig-zag as above, assigning τ_{old} to the decryption of $F_{K_o}(\tau_i)$ under D^{rzz} and τ_i to $F_{K_o}(\tau_i)$, as desired.

Case 3 ($\tau_{\text{old}} \in \mathcal{N}$): This occurs if E^{rzz} does *not* domain-preserve τ_i. This is the case that requires special handling, since if we patch E^{rzz} to map τ_i to $F_{K_o}(\tau_i)$ we may increase the number of domain-preserved points of E^{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}. 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}, p_i^{old}, to be the image of $D^{rzz}_{\text{Emap},\overline{K}}(F_{K_o}(\tau_i))$ 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}.

Note that we do not need special handling of the case that on the ith iteration of KT^{rzz} we assign $E^{rzz}_{\text{Emap},\overline{K}}(\tau_i) = \tau_j$ for some $j > i$. We will change the value of $E^{rzz}_{\text{Emap},\overline{K}}(\tau_j)$ on the ith iteration, but we will fix it in the jth 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 precalculation we need to encode at most $6t$ values—6 times as much as we need to encode to remember $\mathcal{T}$.

9.1 Security of the construction

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 $\mathcal{R}_0, \ldots, \mathcal{R}_t$, such that $\mathcal{R}_0 = \overline{E}$, $\mathcal{R}_i(\tau_j) = F(\tau_j)$ for all $1 \le j \le i \le t$. Each $\mathcal{R}_i$ 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 now state the theorems showing $\mathcal{R}_t$, the cipher $E^{rzz}_{\text{Emap},\overline{K}}$ obtained after the t iterations of KT^{rzz}, is an SPRP. We defer their proofs to the full version. First, we state the information-theoretic step. Intuitively, this theorem shows that if $\overline{E}$ and F are uniformly random permutations, the permutation E^{rzz}_{Emap} resulting from constructing Emap from F and $\overline{E}$ as in KT^{rzz} is also uniformly random.

Theorem 10. *Let $\mathcal{T}$ be a randomly-chosen subset of $\mathcal{D}$, $|\mathcal{T}| = t$, and $t \le d - n$. Let $\mathcal{R}_t$ be a random variable denoting the permutation over $\mathcal{M}$ induced by the RZZ algorithm after all t iterations of KT^{rzz}, instantiated with $\overline{E}$ and F as uniformly random permutations over $\mathcal{M}$ and $\mathcal{D}$, respectively. For any fixed permutation Π on $\mathcal{M}$,*

$$\Pr\left[\, \mathcal{R}_t = \Pi \,\right] = \frac{1}{m!}$$

To complete the proof that $E^{rzz}_{\text{Emap},\overline{K}}$ is an SPRP, we need to transition to the computational setting. Our current proof establishing this uses a reduction that requires exponential time in the worst case, but is efficient in expectation. This is due to the rejection sampling in case 3 of KT^{rzz}, which can take as many as n queries in the worst case but this happens with small probability.

Theorem 11. *Assume that $\overline{E}$ and F are ciphers on domains $\mathcal{M}$ and $\mathcal{D}$, respectively. Let t be a non-zero number, and let $\mathcal{T}$ be a random size t subset $\mathcal{D}$. Let $\mathcal{A}$*

be an SPRP adversary against $E^{rzz}_{Emap,\overline{K}}$ making at most q queries to its oracles. Then the proof gives an adversary $\mathcal{B}$ and an adversary $\mathcal{C}$ such that

$$\mathbf{Adv}^{\mathrm{sprp}}_{E^{rzz}_{Emap,\overline{K}}}(\mathcal{A}(t)) \leq \mathbf{Adv}^{\mathrm{sprp}}_{F}(B) + \mathbf{Adv}^{\mathrm{sprp}}_{\overline{E}}(C)$$

Adversary $\mathcal{B}$ makes at most q queries and runs in time that of $\mathcal{A}$ plus a negligible overhead. Adversary $\mathcal{C}$ runs in expected time $c(q+8t)$ for a small constant c and and makes $q+8t$ queries in expectation.

9.2 Efficiency of Recursive Zig-Zag's *KT*

Now that we know our construction meets the desired security, we turn to analyzing the efficiency of the key transformation KT. Examining the algorithm, we can see that with the exception of the random selection of p_i, the algorithm requires $O(1)$ steps for each iteration, or a time linear in t for the whole calculation. Selecting the p_i's is, however, a bit more involved. As described above, the p_i's are randomly-selected domain-preserved points in $\mathcal{D} \setminus \{\tau_1, \ldots, \tau_{i-1}\}$.

One way of selecting a p_i is to keep picking random points in $\mathcal{D}\setminus\{\tau_1, \ldots, \tau_{i-1}\}$ until a domain-preserved point is found. For a small i, there are many points in $\mathcal{D} \setminus \{\tau_1, \ldots, \tau_{i-1}\}$ and we expect to find an acceptable p_i within very few tries. However, as i approaches $d-n$, for some permutations, the number of domain-preserved elements in $\mathcal{D} \setminus \{\tau_1, \ldots, \tau_{i-1}\}$ may be very small, requiring a large number of tries.

The next theorem shows that we can limit the number of points we need to encrypt in order to generate the p_i's. More specifically, we show that with a high probability, the number of encryptions required for finding the p_i's is linear in t, with a reasonably small factor. Hence, the expected amortized cost of finding each of the p_i is constant.

Theorem 12. *Let X be a random variable (over the probability space defined by the keys of F and $\overline{E}$ and the random choices of the p_i values) whose value is the number of encryptions required to select all the p_i values in the t transformations of KT. Let $d = |\mathcal{D}|$. Then* $\Pr[\, X > 8t \,] \leq e^{-d/8}$.

Proof. If $t > d/8$, where d is the size of the original domain $\mathcal{D}$, we can enumerate $\mathcal{S}_0$ by calculating $\mathcal{R}(x)$ for every $x \in \mathcal{D}$. This requires $d < 8t$ encryptions. Once enumerated, we can calculate $\mathcal{S}_i$ during the generation of the extended permutation without requiring any further encryptions.

For smaller t we look at $s' = d - s$ as a random variable with a hypergeometric distribution whose mean is $E[s'] = d - d^2/(d+n) < d/2$. Hypergeometric distributions are concentrated around the mean. Hence, by Lemma 1, we have

$$\Pr[\, d/4 > s \,] = \Pr[\, d/2 + d/4 < s' \,] < \Pr[\, E[s'] + d/4 < s' \,] < e^{-d/8}$$

Thus, with a high probability, at the i^{th} step in the construction we have a choice of at least $s - i \geq d/4 - t \geq d/8$ elements of $\mathcal{D}$ as candidates for p_i. We can, now, repeatedly pick a random $p_i \in \mathcal{D}$ and check whether $p_i \in \mathcal{S}_i$. Each such try requires one encryption. Because $s - i \geq d/8$ the expected number of encryption required is less than 8. Thus, with a high probability, the expected number of encryption required to select the p_i's is less than $8t$.

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 symposium on Information, computer and communications security*, Singapore, 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*, pages 80–91, Vienna, AT, 2007.
3. Aslan Askarov, Danfeng Zhang, and Andrew C. Myers. Predictive black-box mitigation of timing channels. In Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov, editors, *ACM CCS 10*, pages 297–307, Chicago, Illinois, USA, October 4–8, 2010. ACM Press.
4. Mihir Bellare, Thomas Ristenpart, Phillip Rogaway, and Till Stegers. Format-preserving encryption. In *Selected Areas in Cryptography*, pages 295–312. Springer-Verlag, 2009.
5. Daniel J Bernstein. Cache-timing attacks on AES, 2005. Preprint available at `http://cr.yp.to/papers.html#cachetiming`.
6. John Black and Phillip Rogaway. Ciphers with arbitrary finite domains. In *Topics in Cryptology–CT-RSA 2002*, pages 114–130. Springer Berlin Heidelberg, 2002.
7. David Brumley and Dan Boneh. Remote timing attacks are practical. *Computer Networks*, 48(5):701–716, 2005.
8. V. Chavátal. The tail of the hypergeometric distribution. *Discrete Mathematics*, 25(3):285–287, 1979.
9. David Cock, Qian Ge, Toby C. Murray, and Gernot Heiser. The last mile: An empirical study of timing channels on seL4. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 14*, pages 570–581, Scottsdale, AZ, USA, November 3–7, 2014. ACM Press.
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 *Proceedings of the 20th ACM Conference on Computer and Communications Secuirty (CCS 2013)*, November 2013.
12. Protegrity Inc. Protegrity data protection methods. `http://www.protegrity.com/data-security-platform/#protection-methods`.

13. Moses Liskov, Ronald L Rivest, and David Wagner. Tweakable block ciphers. In *Advances in CryptologyCRYPTO 2002*, pages 31–46. Springer, 2002.
14. 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*, Barcelona, Spain, March 2016.
15. Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B Lee. Last-level cache side-channel attacks are practical. In *2015 IEEE Symposium on Security and Privacy*, pages 605–622, San Jose, CA, US, May 2015.
16. 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 *Proceedings of the 14th conference on USENIX Security Symposium*, 2014.
17. Daniel Luchaup, Thomas Shrimpton, Thomas Ristenpart, and Somesh Jha. Formatted encryption beyond regular languages. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 1292–1303, New York, NY, USA, 2014. ACM.
18. Terence Spies Mihir Bellare, Phillip Rogaway. The ffx mode of operation for format-preserving encryption. `http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/ffx/ffx-spec.pdf`.
19. Dag Arne Osvik, Adi Shamir, and Eran Tromer. Cache attacks and countermeasures: the case of AES. Cryptology ePrint Archive, 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 *Advances in Cryptology – Crypto '13*, LNCS. Springer-Verlag, 2013.
21. Jicheng Shi, Xiang Song, Haibo Chen, and Binyu Zang. Limiting cache-based side-channel in multi-tenant cloud using dynamic page coloring. In *International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pages 194–199, HK, June 2011.
22. Matthew Skala. Hypergeometric tail inequalities: ending the insanity. `http://arxiv.org/pdf/1311.5939v1.pdf`.
23. Inc. Skyhigh Networks. Skyhigh for Salesforce. `https://www.skyhighnetworks.com/product/salesforce-encryption/`.
24. Venkatanathan Varadarajan, Thomas Ristenpart, and Michael Swift. Scheduler-based defenses against cross-VM side-channels. In *Proceedings of the 24th USENIX Security Symposium*, San Diego, CA, US, 2014.
25. HP Security Voltage. Hp format-preserving encryption. `https://www.voltage.com/technology/data-encryption/hp-format-preserving-encryption/`.
26. Zhenghong Wang and Ruby B. Lee. New cache designs for thwarting software cache-based side channel attacks. In *Proceedings of the 34th International Symposium on Computer Architecture*, San Diego, CA, US, 2007.
27. Wikipedia. Tokenization. `https://en.wikipedia.org/wiki/Tokenization_(data_security)`.