Ciphers with Arbitrary Finite Domains

John Black[1] and Phillip Rogaway[2]

[1] Dept. of Computer Science, University of Nevada, Reno NV 89557, USA, jrb@cs.unr.edu, WWW home page: http://www.cs.unr.edu/~jrb

[2] Dept. of Computer Science, University of California at Davis, Davis, CA 95616, USA, rogaway@cs.ucdavis.edu, WWW home page: http://www.cs.ucdavis.edu/~rogaway

Abstract. We explore the problem of enciphering members of a finite set $\mathcal{M}$ where $k = |\mathcal{M}|$ is arbitrary (in particular, it need not be a power of two). We want to achieve this goal starting from a block cipher (which requires a message space of size $N = 2^n$, for some n). We look at a few solutions to this problem, focusing on the case when $\mathcal{M} = [0, k-1]$. We see ciphers with arbitrary domains as a worthwhile primitive in its own right, and as a potentially useful one for making higher-level protocols.

Keywords: Ciphers, Modes of Operation, Provable security, Symmetric Encryption.

1 Introduction

A Motivating Example. Consider the following problem: a company wishes to generate distinct and unpredictable ten-digit credit-card numbers. One way to accomplish this involves keeping a history of all previously-issued numbers. But the company wishes to avoid storing a large amount of sensitive information. Another approach is to use some block cipher E under a randomly-selected key K and then issue credit-card numbers $E_K(0), E_K(1), \cdots$. But the domains of contemporary block ciphers are inconvenient for this problem: this company needs distinct numbers in $[0, 10^{10} - 1]$ but block cipher have a domain $[0, 2^n - 1]$ for some n such as 64 or 128. Is there an elegant solution to this problem?

Enciphering with Arbitrary Domains. More generally now, we have good tools—block ciphers—to encipher points when the message space $\mathcal{M}$ is strings of some particular length, $\mathcal{M} = \{0,1\}^n$. But what if you want to encipher a number between one and a million? Or a point in Z_N or Z_N^*, where N is a 1024-bit number? Or a point from some elliptic-curve group? This paper looks at the question of how to construct ciphers whose domain is *not* $\{0,1\}^n$.

That is, we are interested in how to make a cipher which has some desired but "weird" domain: $F\colon \mathcal{K} \times \mathcal{M} \to \mathcal{M}$ where $\mathcal{K}$ is the key space and $\mathcal{M}$ is the finite message space that we have in mind. A tool from which we may start our construction is a block cipher: a map $E\colon \mathcal{K}' \times \{0,1\}^n \to \{0,1\}^n$ where $\mathcal{K}'$ is the key space and n is the block length. A solution to this problem immediately solves the credit-card problem: for a block cipher $F\colon \mathcal{K}\times[0,10^{10}-1] \to [0,10^{10}-1]$, the company chooses a random $K \in \mathcal{K}$ and issues the (distinct) credit-card numbers $F_K(0), F_K(1), F_K(2), \ldots, F_K(i)$, and has only to remember the last i value used.

Measuring Success. We would like to make clear right away what is the security goal that we are after. Let's do this by way of an example. Suppose that you want to encipher numbers between one and a million: $\mathcal{M} = [1, 10^6]$. Following [2, 7], we imagine two games. In the first game one chooses a random key K from $\mathcal{K}$ and hands to an adversary an oracle $E_K(\cdot)$. In the second game one chooses a random permutation π on $[1, 10^6]$ and hands the adversary an oracle for $\pi(\cdot)$. The adversary should be unable to distinguish these two types of oracles without spending a huge amount of time. Note that the domain is so small that the adversary might well ask for the value of the oracle $f(\cdot) \in \{E_K(\cdot), \pi(\cdot)\}$ at *every* point in the domain. This shouldn't help the adversary win. So, for example, if the adversary asks the value of $E_K(\cdot)$ at all points except 1 and 2 (a total of 10^6-2 points), then the adversary will know what are the two "missing" numbers, c_1 and c_2, but the adversary won't be able to ascertain if $E_K(1) = c_1$ and $E_K(2) = c_2$, or if $E_K(1) = c_2$ and $E_K(2) = c_1$, instead.

Our Contributions. Though the problem of enciphering on an arbitrary domain has been considered before [13], here we draw attention to this problem and give the first rigorous treatment, providing a few solutions together with their analyses. Our solutions focus on the case in which the message space is $\mathcal{M} = [0, k-1]$, though we sketch extensions to some other message spaces, like Z^*_{pq} and common elliptic-curve groups.

Our first method assumes that we have a block cipher E that acts on $N = 2^n$ points, where $N \geq k$. To encipher $\mathcal{M} = [0, k-1]$ one just enciphers these points with block cipher E and uses the ordering of $E_K(0)$, $E_K(1)$, up to $E_K(k-1)$ to name the desired permutation on $[0, k-1]$. This method is computationally reasonable only for small k, such as $k < 2^{30}$.

A second method, similar to known techniques used in other settings, enciphers a message $m \in \mathcal{M}$ by repeatedly applying the block cipher, starting at m, until one gets back to a point in $\mathcal{M}$. (Assume once again that $N \geq k$.) This method is good if $\mathcal{M}$ is "dense" in the domain of the block cipher, $\{0,1\}^n$. So, for example, one can use this method to encipher a string in Z_N, where N is a 1024-bit number, using a block cipher with block length of 1024 bits. (A block cipher with a long block length, like this, can be constructed from a "standard" block cipher by following works like [3, 9, 11].) This construction has been suggested before [13]; our main contribution here is the analysis of the construction.

A final method which we look at chooses an a, b where $ab \geq k$ and performs a Feistel construction on the message m, but uses a left-hand side in Z_a and a right-hand side in Z_b. Our analysis of this is an adaptation of Luby and Rackoff's [9]. This method can be quite efficient, though the proven bounds are weak when the message space is small (eg, $k < 2^{128}$).

With each of our ciphers we provide a deciphering algorithm, though this may not be required in all domains (eg, in our credit-card example above).

Note that the three methods above solve our problem for small and large domains, but there is a gap which remains: intermediate-sized values where our first method requires too much space and time, and our second method requires too many block-cipher invocations, and our third method may work but the bound is too weak. This gap occurs roughly from $k = 2^{30}$ up to about $k = 2^{60}$, depending on your point of view. Our credit-card example ($k = 10^{10} \approx 2^{33.2}$) falls into this gap. This problem remains open.

Why Ciphers on Non-Standard Sets? Popular books on cryptography speak of enciphering the points in the message space $\mathcal{M}$, whatever that message space may be, but few seem to have thought much about how to actually do this when the message space is something other than a set of bit strings, often of one particular length. This omission is no doubt due to the fact that it is usually fine to embed the desired message space into a larger one, using some padding method, and then apply a standard construction to encipher in the larger space. For example, suppose you want to encipher a random number m between one and a million. Your tool is a 128-bit block cipher E. You could encode m as a 128-bit string M by writing m using 20 bits, prepending 108 zero-bits, and computing $C = E_K(M)$. Ignoring the fact that the ciphertext C "wastes" 108 bits, this method is usually fine. But not always.

One problem with the method above is that it allows one to tell if a candidate key K' might have been used to produce C. To illustrate the issue, suppose that the key space is small, say $|\mathcal{K}| = 2^{30}$. Suppose the adversary sees a point $C = E_K(M)$. Then the adversary has everything she needs to decrypt ciphertext $C = E_K(M)$: she just tries all keys $K' \in \mathcal{K}$ until she finds one for which $E_{K'}^{-1}(C)$ begins with 108 zeros. This is almost certainly the right key. The objection that "we shouldn't have used a small key space" is not a productive one if the point of our efforts was to make due with a small key space.

If we had used a cipher with message space $\mathcal{M} = [1, 10^6]$ we would not have had this problem. Every ciphertext C, under every possible key K, would correspond to a valid message M. The ciphertext would reveal nothing about which key had been used.

Of course there are several other solutions to the problem we have described, but many of them have difficulties of their own. Suppose, for example, that one pads with random bits instead of zero bits. This is better, but still not perfect: in particular, an adversary can tell that a candidate key K' could not have been used to encipher M if decrypting C under K' yields a final 20 bits whose decimal value exceeds 1,000,000. If one had 1,000 ciphertexts of random plaintexts enciphered in the manner we have described, the adversary could, once again, usually determine the correct key.

As a more realistic example related to that above, consider the Bellovin-Merritt "EKE" protocol [4]. This entity-authentication protocol is designed to defeat password-guessing attacks. The protocol involves encrypting, under a possibly weak password K, a string $g^x \bmod p$, where p is a large prime number and g is a generator of Z_p^*. In this context it is crucial that from the resulting ciphertext C one can not ascertain if a candidate password K' could possibly have produced the ciphertext C. This can be easily and efficiently done by enciphering with message space $\mathcal{M} = Z_p^*$. Ordinary encryption methods won't work.

Another problem with ciphertext-expansion occurs when we are constrained by an existing record format: suppose we wish to encrypt a set of fields in a database, but the cost of changing the record size is prohibitive. Using a cipher whose domain is the set of values for the existing fields allows some measure of added security without requiring a complete restructuring of the database. And if the data have additional restrictions beyond size (eg, the fields must contain printable characters), we can further restrict the domain as needed.

In addition to these (modest) applications, the question is interesting from a theoretical standpoint: how can we construct new ciphers from existing ones? In particular, can we construct ciphers with arbitrary domains without resorting to creating new ciphers from scratch? It certainly "feels" like there should be a good way to construct a block cipher on 32 bits given a block cipher on 64 bits, but, even for this case, no one knows how to do this in a practical manner with good security bounds.

Related Work. We assume that one has in hand a good block cipher for any desired block length. Since "standard" block ciphers come only in "convenient" block lengths, such as $n = 128$, here are some ways that one might create a block cipher for some non-standard block length. First, one could construct the block cipher from scratch. But it is probably better to start with a well-studied primitive like SHA-1 or AES. These could then be used within a balanced Feistel network [14], which creates a block cipher for any (even) block length $2n$, starting with something that behaves as a pseudorandom function (PRF) from n bits to n bits. Luby and Rackoff [9] give quantitative bounds on the efficacy of this construction (when using three and four rounds), and their work has spawned much related analysis, too. Naor and Reingold [11] provide a different construction which extends a block cipher on n bits to a block cipher on $2ni$ bits, for any $i \geq 1$. A variation on their construction due to Patel, Ramzan and Sundaram [12] yields a cipher on ni bits for any $i \geq 1$. Lucks [10] generalizes Luby-Rackoff to consider a three-round unbalanced Feistel network, using hash functions for round functions. This yields a block cipher for any given length N starting with a PRF from r bits to ℓ bits and another from ℓ bits to r bits, where $\ell + r = N$. Starting from an n-bit block cipher, Bellare and Rogaway [3] construct and analyze a length-preserving cipher with domain $\{0,1\}^{\geq n}$. This is something more than making a block cipher on arbitrary $N \geq n$ bits. Anderson and Biham [1] provide two constructions for a block cipher (BEAR and LION) which use a hash function and a stream cipher. This again uses an unbalanced Feistel network.

It is unclear how to make any of the constructions above apply to message spaces which are not sets of strings. Probably several of the constructions can modified, and in multiple ways, to deal with a message space $\mathcal{M} = [0, k-1]$, or with other message spaces.

The Hasty Pudding Cipher of Schroeppel and Orman [13] is a block cipher which works on any domain $[0, k-1]$. They use what is essentially "Method 2," internally iterating the cipher until a proper domain point is reached. Schroeppel believes that the idea underlying this method dates back to the rotor machines used in the early 1900's.

Our notion of a pseudorandom function is due to Goldreich, Goldwasser and Micali [6]. Pseudorandom permutations are defined and constructed by Luby and Rackoff [9]. We use the adaptation of these notions to deal with finite objects, which first appears in Bellare, Kilian and Rogaway [2].

2 Preliminaries

Notation. If A and B are sets then $\mathrm{Rand}(A, B)$ is the set of all functions from A to B. If A or B is a positive number, n, then the corresponding set is $[0, n-1]$. We write $\mathrm{Perm}(A)$ to denote the set of all permutations on the set A and if n is a positive number then the set is assumed to be $[0, n-1]$. By $x \stackrel{R}{\leftarrow} A$ we denote the experiment of choosing a random element from A.

A function family is a multiset $F = \{f \colon A \to B\}$, where $A, B \subseteq \{0,1\}^*$. Each element $f \in F$ has a name K, where $K \in \mathsf{Key}$. So, equivalently, a function family F is a function $F \colon \mathsf{Key} \times A \to B$. We call A the domain of F and B the range of F. The first argument to F will be written as a subscript. A cipher is a function family $F \colon \mathsf{Key} \times A \to A$ where $F_K(\cdot)$ is always a permutation; a block cipher is a function family $F \colon \mathsf{Key} \times \{0,1\}^n \to \{0,1\}^n$ where $F_K(\cdot)$ is always a permutation. An ideal block cipher is a block cipher in which each permutation on $\{0,1\}^n$ is realized by exactly one $K \in \mathsf{Key}$.

An adversary is an algorithm with an oracle. The oracle computes some function. We write $A^{f(\cdot)}$ to indicate an adversary A with oracle $f(\cdot)$. Adversaries are assumed to never ask a query outside the domain of the oracle, and to never repeat a query.

Let $F \colon \mathsf{Key} \times A \to B$ be a function family and let $\mathcal{A}$ be an adversary. In this paper, we measure security as the maximum advantage obtainable by some adversary; we use the following statistical measures:

$$\mathbf{Adv}_F^{\mathrm{prf}}(\mathcal{A}) \stackrel{\mathrm{def}}{=} \Pr[f \stackrel{R}{\leftarrow} F \colon \mathcal{A}^{f(\cdot)} = 1] - \Pr[R \stackrel{R}{\leftarrow} \mathrm{Rand}(A, B) \colon \mathcal{A}^{R(\cdot)} = 1] \;,$$

and when $A = B$

$$\mathbf{Adv}_F^{\mathrm{prp}}(\mathcal{A}) \stackrel{\mathrm{def}}{=} \Pr[f \stackrel{R}{\leftarrow} F \colon \mathcal{A}^{f(\cdot)} = 1] - \Pr[\pi \stackrel{R}{\leftarrow} \mathrm{Perm}(A) \colon \mathcal{A}^{\pi(\cdot)} = 1] \;.$$

Useful Facts. It is often convenient to replace random permutations with random functions, or vice versa. The following proposition lets us easily do this. For a proof see Proposition 2.5 in [2].

Lemma 1. [PRF/PRP Switching] *Fix $n \geq 1$. Let $\mathcal{A}$ be an adversary that asks at most p queries. Then*

$$\left| \Pr[\pi \stackrel{R}{\leftarrow} \mathrm{Perm}(n) \colon \mathcal{A}^{\pi(\cdot)} = 1] \;-\; \Pr[\rho \stackrel{R}{\leftarrow} \mathrm{Rand}(n, n) \colon \mathcal{A}^{\rho(\cdot)} = 1] \right| \;\leq\; p^2/2^{n+1}.$$

Algorithm Init_Px_K
  for $j \leftarrow 0$ **to** $k-1$ **do** $I_j \leftarrow E_K(j)$
  for $j \leftarrow 0$ **to** $k-1$ **do** $J_j \leftarrow \text{Ord}(I_j, \{I_j\}_{j\in[0,k-1]})$
  for $j \leftarrow 0$ **to** $k-1$ **do** $L_{J_j} \leftarrow j$

Algorithm $\text{Px}_K(m)$
  return J_m

Algorithm $\text{Px}_K^{-1}(m)$
  return L_m

Fig. 1. *Algorithms for the Prefix Cipher. First the initialization algorithm* Init_Px_K *is run. Then encipher with* $\text{Px}_K(m)$ *and decipher with* $\text{Px}_K^{-1}(m)$.

3 Method 1: Prefix Cipher

Fix some integer k and let $\mathcal{M}$ be the set $[0, k-1]$. Our goal is to build a cipher with domain $\mathcal{M}$.

Our first approach is a simple, practical method for small values of k. We name this cipher Px. Our cipher will use some existing block cipher E with keyspace $\mathcal{K}$ and whose domain is a superset of $\mathcal{M}$. The key space for Px will also be $\mathcal{K}$. To compute $\text{Px}_K(m)$ for some $m \in \mathcal{M}$ and $K \in \mathcal{K}$ we first compute the tuple

$$I = (E_K(0)\ E_K(1)\ \cdots\ E_K(k-1)).$$

Since each element of I is a distinct string, we may replace each element in I with its ordinal position (starting from zero) to produce tuple J. And now to encipher any $m \in \mathcal{M}$ we compute $\text{Px}_K(m)$ as simply the m-th component of J (again counting from zero). The enciphering and deciphering algorithms are given in Figure 1.

Example. Suppose we wish to encipher $\mathcal{M} = \{0, 1, 2, 3, 4\}$. We choose some random key K for some block cipher E. Let's assume E is an 8-bit ideal block cipher; therefore E_K is a uniformly chosen random permutation on $[0, 255]$. Next we encipher each element of $\mathcal{M}$. Let's say $E_K(0) = 166$, $E_K(1) = 6$, $E_K(2) = 130$, $E_K(3) = 201$, and $E_K(4) = 78$. So our tuple I is (166 6 130 201 78) and J is (3 0 2 4 1). We are now ready to encipher any $m \in \mathcal{M}$: we return the m-th element from J, counting from zero. For example we encipher 0 as 3, and 1 as 0, etc..

Analysis. Under the assumption that our underlying block cipher E is ideal, I is equally likely to be any of the permutations on $\mathcal{M}$. The proof of this fact is trivial and is omitted. The method remains good when E is secure in the sense of a PRP. The argument is standard and is omitted.

Practical Considerations. Enciphering and deciphering are constant-time operations. The cost here is $O(k)$ time and space used in the initialization step. This clearly means that this method is practical only for small values of k. A further practical consideration is that, although this initialization is a one-time cost, it results in a table of sensitive data which must be stored somewhere.

Algorithm $\mathrm{Cy}_K(m)$	**Algorithm** $\mathrm{Cy}_K^{-1}(m)$
$c \leftarrow E_K(m)$	$c \leftarrow E_K^{-1}(m)$
if $c \in \mathcal{M}$ **return** c	**if** $c \in \mathcal{M}$ **return** c
else return $\mathrm{Cy}_K(c)$	**else return** $\mathrm{Cy}_K^{-1}(c)$

Fig. 2. *Algorithms for the Cycle-Walking Cipher. We encipher with* $\mathrm{Cy}_K(\cdot)$ *and decipher with* $\mathrm{Cy}_K^{-1}(\cdot)$.

4 Method 2: Cycle-Walking Cipher

This next method uses a block cipher whose domain is larger than $\mathcal{M}$, and then handles those cases where a point is out of range. Again we fix an integer k, let $\mathcal{M}$ be the set $[0, k-1]$, and devise a method to encipher $\mathcal{M}$.

Let N be the smallest power of 2 larger or equal to k, let n be $\lg N$, and let $E_K(\cdot)$ be an n-bit block cipher. We construct the block cipher Cy_K on the set $\mathcal{M}$ by computing $t = E_K(m)$ and iterating if $c \notin \mathcal{M}$. The enciphering and deciphering algorithms are shown in Figure 2.

Example. Let $\mathcal{M} = [0, 10^6]$. Then $N = 2^{20}$ and so $n = 20$. We use some known method to build a 20-bit block cipher $E_K(\cdot)$ on the set $\mathcal{T} = [0, 2^{20}-1]$. Now suppose we wish to encipher the point $m = 314159$; we compute $c_1 = E_K(314159)$ which yields some number in $\mathcal{T}$, say 1040401. Since $c_1 \notin \mathcal{M}$, we iterate by computing $c_2 = E_K(1040401)$ which is, say, 1729. Since $c_2 \in \mathcal{M}$, we output 1729 as $\mathrm{Cy}_K(314159)$. Decipherment is simply the reverse of this procedure.

Analysis. Let's view the permutation $E_K(\cdot)$ as a family of cycles: any point $m \in \mathcal{M}$ lies on some cycle and repeated applications of $E_K(\cdot)$ can be viewed as a particle walking along the cycle, starting at m. In fact, we can now think of our construction as follows: to encipher any point $m \in \mathcal{M}$ walk along the cycle containing m until you encounter some point $c \in \mathcal{M}$. Then $c = \mathrm{Cy}_K(m)$. Of course this method assumes that one can efficiently test for membership in $\mathcal{M}$. This is trivial for our case when $\mathcal{M} = [0, k-1]$, but might not be for other sets.

Now we may easily see that $\mathrm{Cy}_K(\cdot)$ is well-defined: given any point $m \in \mathcal{M}$ if we apply $E_K(\cdot)$ enough times, we will arrive at a point in $\mathcal{M}$. This is because walking on m's cycle must eventually arrive back at *some* point in $\mathcal{M}$, even if that point is m itself. We can also see that $\mathrm{Cy}_K(\cdot)$ is invertible since inverting $\mathrm{Cy}_K(m)$ is equivalent to walking *backwards* on m's cycle until finding some element in $\mathcal{M}$. Therefore, we know $\mathrm{Cy}_K(\cdot)$ is a permutation on $\mathcal{M}$. However the question arises, "how much security do we lose in deriving this permutation?" The fortunate answer is, "nothing."

Theorem 1. [Security of Cycle-Walking Cipher] *Fix* $k \geq 1$ *and let* $\mathcal{M} = [0, k-1]$. *Let* $E_K(\cdot)$ *be an ideal block cipher on the set* $\mathcal{T}$ *where* $\mathcal{M} \subseteq \mathcal{T}$. *Choose a key* K *uniformly at random and then construct* $\mathrm{Cy}_K(\cdot)$ *using* $E_K(\cdot)$. *Then* $\mathrm{Cy}_K(\cdot)$ *is a uniform random permutation on* $\mathcal{M}$.

Proof. Fix some permutation π on the set $\mathcal{M}$. We will show that an equal number of keys K will give rise to π; this will imply the theorem.

We proceed by induction, showing that the number of permutations on $\{0, \ldots, k-1, x\}$ which give rise under our construction to π is constant. Since $\mathcal{M} \subseteq \mathcal{T}$ we can repeatedly add all elements $x \in \mathcal{T} - \mathcal{M}$ while maintaining that the number of permutations which give rise to π is constant.

Decompose π into r cycles of lengths $l_1, l_2, \cdots, l_r$. We count the number of ways to insert the new element x. There are l_i ways to insert x into the ith orbit corresponding to the ith cycle, and one way to insert x into a new orbit of its own (ie, the permutation which fixes x). Therefore there are $\sum_{i=1}^{r} l_i + 1 = k$ ways to add element x to π yielding a permutation which will give rise to π by repeated iterations. This holds no matter what π we choose.

Let $|\mathcal{T}| = t$. Then by induction we see that there are exactly $\prod_{i=k}^{t} i$ keys K under which our construction reduces $E_K(\cdot)$ to π.

Similar to the Prefix Cipher, our construction has retained all of the security of the underlying block cipher.

Theorem 1 is an information-theoretic result. Passing to the corresponding complexity-theoretic result is standard. Because no security is lost in the information-theoretic setting, and because we apply E an expected two times (or fewer), an adversary's maximal advantage to distinguish $E_K(\cdot)$ from a random permutation of Z_{2^n} in expected time $2t$ approximately upper bounds an adversary's maximal advantage to distinguish $\mathrm{Cy}_K(\cdot)$ from a random permutation on $\mathcal{M}$ in time t.

5 Method 3: Generalized-Feistel Cipher

Our final method works as follows: we decompose all the numbers in $\mathcal{M}$ into pairs of "similarly sized" numbers and then apply the well-known Feistel construction [14] to produce a cipher. Again we fix an integer k, let $\mathcal{M}$ be the set $[0, k-1]$, and devise a method to encipher $\mathcal{M}$.

We call our cipher $\mathrm{Fe}[r, a, b]$ where r is the number of rounds we use in our Feistel network and a and b are positive numbers such that $ab \geq k$. We use a and b to decompose any $m \in \mathcal{M}$ into two numbers for use as the inputs into the network. Within the network we use r random functions $F_1, \ldots, F_r$ whose ranges contain $\mathcal{M}$. The algorithms to encipher and decipher are given in Figure 3. Notice that if using the Feistel construction results in a number not in $\mathcal{M}$, we iterate just as we did for the Cycle-Walking Cipher.

Example. In order to specify some particular $\mathrm{Fe}[r, a, b]_K(\cdot)$ we must specify the numbers a and b, the number of Feistel rounds r, and the choice of underlying functions $F_1, \cdots, F_r$ we will use.

As a concrete example, let's take $k = 2^{35}$, $r = 3$, and $a = 185360$ and $b = 185368$ (methods for finding a and b will be discussed later). Note that $ab \geq k$ as required. Since ab is 74112 larger than k, our Feistel construction will be on the set $\mathcal{M}' = [0, (2^{35} - 1) + 74112]$, meaning there are 74112 values

```
Algorithm Fe[r, a, b]_K(m)
    c ← fe[r, a, b]_K(m)
    if c ∈ M return c
    else return Fe[r, a, b]_K(c)

Algorithm fe[r, a, b]_K(m)
    L ← m mod a;   R ← ⌊m/a⌋
    for j ← 1 to r do
        if (j is odd) then tmp ← (L + F_j(R)) mod a
        else tmp ← (L + F_j(R)) mod b
        L ← R;   R ← tmp
    if (r is odd) then return aL + R
    else return aR + L
```

```
Algorithm Fe[r, a, b]_K^{-1}(m)
    c ← fe[r, a, b]_K^{-1}(m)
    if c ∈ M return c
    else return Fe[r, a, b]_K^{-1}(c)

Algorithm fe[r, a, b]_K^{-1}(m)
    if (r is odd) then R ← m mod a;   L ← ⌊m/a⌋
    else L ← m mod a;   R ← ⌊m/a⌋
    for j ← r to 1 do
        if (j is odd) then tmp ← (R − F_j(L)) mod a
        else tmp ← (R − F_j(L)) mod b
        R ← L;   L ← tmp
    return aR + L
```

Fig. 3. *Algorithms for the Generalized-Feistel Cipher. We encipher with* $\mathrm{Fe}[r,a,b]_K(\cdot)$ *and decipher with* $\mathrm{Fe}[r,a,b]_K^{-1}(\cdot)$*. Here* a *and* b *are the numbers used to bijectively map all* $m \in \mathcal{M}$ *into* L*, and* R*, and* r *is the number of rounds of Feistel we will apply. The key* K *is implicitly used to select the* r *functions* $F_1, \ldots, F_r$*.*

which are in $\mathcal{M}' - \mathcal{M}$ for which we will have to iterate (just as we did for the Cycle-Walking Cipher). Let's use DES with independent keys as our underlying PRFs. DES is a 64-bit cipher which uses a 56-bit key; we will regard the 64-bit strings on which DES operates as integers in the range $[0, 2^{64} - 1]$ in the natural way. We need three PRFs so our key $K = K_1 \parallel K_2 \parallel K_3$ will be $3 \times 56 = 168$ bits. Now to compute $\mathrm{Fe}[3, 185360, 185368](m)$ we compute $L = m \bmod 185360$, and $R = \lfloor m/185360 \rfloor$, and then perform three rounds of Feistel using $\mathrm{DES}_{K_1}(\cdot)$, $\mathrm{DES}_{K_2}(\cdot)$, and $\mathrm{DES}_{K_3}(\cdot)$ as our underlying PRFs. The first round results in $L \leftarrow \lfloor m/185360 \rfloor$ and $R \leftarrow (m \bmod 185360 + DES_{K_1}(\lfloor m/185360 \rfloor)) \bmod 185360$, and so on.

Analysis. First we note that $\mathrm{Fe}[r,a,b](\cdot)$ is a permutation: it is well-known that the Feistel construction produces a permutation, and we showed previously that iterating any permutation is a permutation. We now analyze the how good is this Generalized-Feistel Cipher for the three-round case.

Assuming the underlying functions F_1, F_2, and F_3 used in our construction are *truly random* functions, we will compare how close $\mathrm{Fe}[3,a,b](\cdot)$ is to a truly random permutation. Passing to the complexity-theoretic setting is then standard, and therefore omitted.

Theorem 2. [Security of Generalized-Feistel Cipher] *Fix $k \geq 1$ and let $\mathcal{M} = [0, k-1]$. Fix two numbers $a, b > 0$ such that $ab \geq k$. Let $\Delta = ab - k$. Fix an n such that $2^n > a$ and $2^n > b$. Let D be an adversary which asks q queries of her oracle. Then*

$$\begin{aligned}\mathbf{Adv}^{\mathrm{prf}}_{\mathrm{Fe}}(D) &= \Pr[F_1, F_2, F_3 \stackrel{R}{\leftarrow} \mathrm{Rand}(2^n, 2^n):\ D^{\mathrm{Fe}[3,a,b](\cdot)} = 1] \\ &\qquad - \Pr[\rho \stackrel{R}{\leftarrow} \mathrm{Rand}(k,k):\ D^{\rho(\cdot)} = 1] \\ &\leq \frac{(q+\Delta)^2}{2^{n+1}} \left(\lceil 2^n/a \rceil + \lceil 2^n/b \rceil\right).\end{aligned}$$

The proof is an adaptation of Luby's analysis from Lecture 13 of [8], which is in-turn based on [9]. It can be found in Appendix A.

Finally, we must adjust this bound to account for the fact that we have compared $\mathrm{Fe}[3,a,b]_K(\cdot)$ with a random function instead of a random permutation. We can invoke Lemma 1 which gives us a final bound quantifying the quality of our construction:

$$\begin{aligned}\mathbf{Adv}^{\mathrm{prp}}_{\mathrm{Fe}}(D) &= \Pr[F_1, F_2, F_3 \stackrel{R}{\leftarrow} \mathrm{Rand}(2^n, 2^n):\ D^{\mathrm{Fe}[3,a,b](\cdot)} = 1] \\ &\qquad - \Pr[\pi \stackrel{R}{\leftarrow} \mathrm{Perm}(k):\ D^{\pi(\cdot)} = 1] \\ &\leq \frac{(q+\Delta)^2 + q^2}{2^{n+1}} \left(\lceil 2^n/a \rceil + \lceil 2^n/b \rceil\right).\end{aligned}$$

6 Discussion

Prefix Cipher. Our first method, the Prefix Cipher, is useful only for suitably small k. Since enciphering one point requires enciphering all k points in $[0, k-1]$, many applications would find this prohibitively expensive for all but fairly small values of k.

Cycle-Walking Cipher. Our second method, the Cycle-Walking Cipher, can be quite practical. If k is just smaller than some power of 2, the number of points we have to "walk through" during any given encipherment is correspondingly small. In the worst case, however, k is one larger than a power of 2, and (with extremely bad luck) might require k calls to the underlying block cipher to encipher just one point. But if the underlying block cipher is good we require, in the worst case, an expected two calls to it in order to encipher and decipher any point.

Generalized-Feistel Cipher. To get the best bound we should select a and b such that these numbers are somewhat close together and such that $\Delta = ab - k$ is small. One obvious technique is to try numbers near $\sqrt{k}$; for example, taking $a = b = \lceil\sqrt{k}\rceil$ means that $ab - k$ will never be more than $2\sqrt{k} + 1$. But often one can do better.

Another way to improve the bound is to ensure n is suitably large. The "tail effects" spoken of in the proof are diminished as n grows (because as 2^n gets larger $\lceil 2^n/a\rceil/2^n$ gets closer to $1/a$).

The One-Off Construction. Another method, not mentioned above, works well for domains which are one element larger than a domain we can accommodate efficiently. Say we have a cipher E with domain $[0, k-1]$ and we wish to construct a cipher E' with domain $[0, k]$. We choose a key $K' = \{K, r\}$ for E' by choosing a key K for E and a random number $r \in [0, k]$. We then compute $E'_{K'}(X)$ as follows:

$$E'_{K'}(X) = \begin{cases} r & \text{if } X = k \\ k & \text{if } X = E_K^{-1}(r) \\ E_K(X) & \text{otherwise} \end{cases}$$

The security of this construction is tightly related to the security of E and the method for selecting r. The analysis is omitted.

Of course we can use this method to repeatedly extend the domain of any cipher to the size of choice, but for most settings it is impractical to do this more than a few times. A typical method for generating r would be to take $r = E_{K^*}(0) \bmod (k+1)$ where K^* is a new randomly-selected key. The "tail effect" here is not too bad, but will cause a rapid deterioration of the security bound when used too often. Also, the scheme begins to become quite inefficient when we extend the domain in this way too many times.

Other Domains. Though we have spoken in terms of the domain $[0, k-1]$ the same methods work for other domains, too. For example, to encipher in Z_N^*, where $N = pq$ is a 1024-bit product of two primes, one can use either cycle-walking or the generalized-Feistel construction, iterating in the highly unlikely event that a point is in Z_N but not in Z_N^*.

We may also use our methods to encipher points from an elliptic curve group (EC group). There are well-known "compact" representations of the points in EC groups, and these representations form our starting point. For example, one finds in [5] simple algorithms to compress the representation of a point in an EC group. Consider the EC group G over the field F_q where q is either a power of two or a prime. Then any point $(x, y) \in G$ may be represented as a member of F_q together with a single bit. Let's consider first the case where $q = 2^m$ with $m > 0$. The Hasse theorem (see [5], page 8) guarantees at least $d(r) = r + 1 - 2\sqrt{r}$ points in G. Since it is possible to represent any point in G with $m+1$ bits and it is also possible to efficiently test for membership in G, we could use the cycle-walking construction over a 2^{m+1}-bit cipher. The expected number of invocations of this cipher to encipher a point in G is then $2^{m+1}/d(2^m) \approx 2$.

If q is instead a prime p, we can represent any point in G as a number $x \in [0, p-1]$ and a single bit y. We may again use any of our methods to encipher these $2p$ points. Here the Hasse theorem ([5], page 7) guarantees at least $d(p)$ points in G and once again an efficient test for membership in G exists. Therefore we may use the cycle-walking construction over some $\lceil \lg\ 2p \rceil$-bit cipher. However if $2p$ is not close to a power of 2, we may wish to instead use the generalized-Feistel construction.

Open Problems. As mentioned already, we have not provided any construction which works well (and provably so) for intermediate-sized values of k. For example, suppose you are given an ideal block cipher Π on 128-bit strings, and you want to approximate a random permutation π on, say, 40-bit strings. Probably enough rounds of Feistel work, but remember that our security goal is that even if an adversary inquires about all 2^{40} points, still she should be unable to distinguish π from a random permutation on 40 bits. Known bounds are not nearly so strong. Of course the prefix method works, but spending 2^{40} time and space to encipher the first point is not practical.

Acknowledgments

Special thanks to Richard Schroeppel who made many useful comments on an earlier draft. Thanks also to Mihir Bellare, David McGrew, and Silvio Micali for their helpful comments. This paper was written while Rogaway was on leave of absence from UC Davis, visiting the Department of Computer Science, Faculty of Science, Chiang Mai University. This work was supported under NSF CAREER award CCR-9624560, and by a generous gift from Cisco Systems.

References

1. Anderson, R., and Biham, E. Two practical and provably secure block ciphers: BEAR and LION. In *Fast Software Encryption* (1996), vol. 1039 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 114–120.
2. Bellare, M., Kilian, J., and Rogaway, P. The security of the cipher block chaining message authentication code. *Journal of Computer and System Sciences 61*, 3 (2000), 362–399. Earlier version in CRYPTO '94. See www.cs.ucdavis.edu/~rogaway.
3. Bellare, M., and Rogaway, P. On the construction of variable-input-length ciphers. In *Fast Software Encryption* (1999), vol. 1636 of *Lecture Notes in Computer Science*, Springer-Verlag. See www.cs.ucdavis.edu/~rogaway.
4. Bellovin, S., and Merritt, M. Encrypted key exchange: password-based protocols secure against dictionary attacks. In *1992 IEEE Computer Society Symposium on Research in Security and Privacy* (1992), IEEE Computer Society Press, pp. 72–84.
5. Certicom Research. *Standards for efficient cryptography, SEC1: Elliptic curve cryptography*, version 1, Sept. 2000. Available on-line at www.secg.org.
6. Goldreich, O., Goldwasser, S., and Micali, S. How to construct random functions. *Journal of the ACM 33*, 4 (1986), 210–217.

7. Goldwasser, S., Micali, S., and Rivest, R. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal of Computing 17*, 2 (Apr. 1988), 281–308.
8. Luby, M. *Pseudorandomness and cryptographic applications.* Princeton University Press, Princeton, New Jersey, 1996.
9. Luby, M., and Rackoff, C. How to construct pseudorandom permutations from pseudorandom functions. *SIAM Journal of Computing 17*, 2 (Apr. 1988).
10. Lucks, S. Faster Luby-Rackoff ciphers. In *Fast Software Encryption* (1996), vol. 1039 of *Lecture Notes in Computer Science*, Springer-Verlag.
11. Naor, M., and Reingold, O. On the construction of pseudorandom permutations: Luby-Rackoff revisited. *Journal of Cryptology 12*, 1 (1999), 29–66.
12. Patel, S., Ramzan, Z., and Sundaram, G. Towards making Luby-Rackoff ciphers optimal and practical. In *Fast Software Encryption* (1999), vol. 1636 of *Lecture Notes in Computer Science*, Springer-Verlag.
13. Schroeppel, R., and Orman, H. Introduction to the hasty pudding cipher. In Proceedings from the First Advanced Encryption Standard Candidate Conference, National Institute of Standards and Technology, Aug. 1998. See `http://www.cs.arizona.edu/~rcs/hpc/`.
14. Smith, J. L. The design of Lucifer: A cryptographic device for data communications. Tech. Rep. IBM Research Report RC 3326, IBM T.J. Watson Research Center, Yorktown Heights, N.Y., 10598, U.S.A., Apr. 1971.

A Proof of Theorem 2

Proof. To simplify the exposition, we will initially assume that $k = ab$. In other words, that no iterating is required to compute $\mathrm{Fe}[3, a, b]_K(\cdot)$. Once we establish the result in this setting, we can make some minor changes to get the general result.

We begin by defining a couple of games. Let us call "Game Fe" the game in which we choose three random functions $F_1, F_2, F_3 \leftarrow \mathrm{Rand}(2^n, 2^n)$ and then answer D's queries according to $\mathrm{Fe}[3, a, b](\cdot)$ using F_1, F_2, and F_3 as our underlying functions. Let us call "Game Rn" the game in which we choose a random function $\rho \in \mathrm{Rand}(k, k)$ and then answer D's queries according to $\rho(\cdot)$. Let's denote by P_{Fe} the probability that D outputs 1 in Game Fe, and denote by P_{Rn} the probability that D outputs 1 in Game Rn. We are trying to show that

$$P_{\mathrm{Fe}} - P_{\mathrm{Rn}} \leq \frac{(q + ab - k)^2}{2^{n+1}} \left(\lceil 2^n/a \rceil + \lceil 2^n/b \rceil\right).$$

Without loss of generality, assume D never repeats a query. We begin by describing a new game called "Game B′". Game B′ will look the same to adversary D as Game Fe, but Game B′ will be played completely differently. Instead of choosing three random functions F_1, F_2, F_3, we'll choose only some random numbers $x_1, \ldots, x_q$, $y_1, \ldots, y_q$, and $z_1, \ldots, z_q$. Each of these numbers is in $[0, 2^n - 1]$. The *only* random choices we will make in playing game B' is in the choice of the x_i, y_i, and z_i. We describe Game B′ in Figure 4. It is played as follows: first choose random numbers $x_1, \ldots, x_q$, $y_1, \ldots, y_q$, and $z_1, \ldots, z_q$. Now answer the i-th query with $a\beta_i + \gamma_i$, where β_i and γ_i are described in the figure.

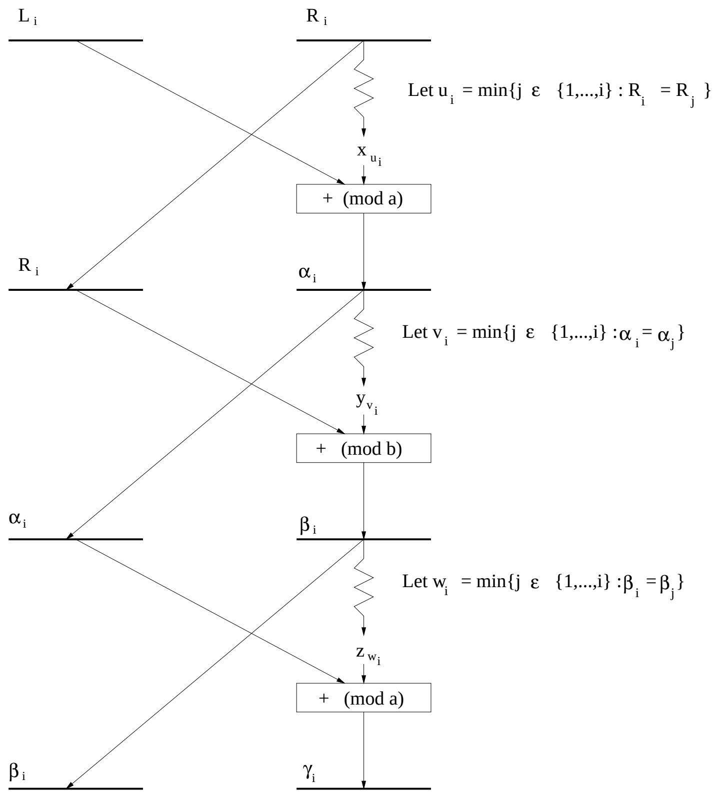

Fig. 4. *Game* B′. *This game is identical, as far as the adversary can tell, to Game Fe. Begin by choosing* $x_1, \ldots, x_q$, $y_1, \ldots, y_q$, *and* $z_1, \ldots, z_q$ *at random. Then answer the i-th query,* L_i, R_i, *by* β_i, γ_i, *computed as in the figure.*

It should be obvious that Game B′ is the same, as far as the adversary can see, to Game Fe. Thus $P_{\mathrm{Fe}} = \Pr[D^{B'} = 1]$.

We now modify Game B′ to a Game B which is identical, from the adversary's point of view, to Game B′ (and therefore to Game Fe). This modification is unusual: we will subtract R_{v_i} from the second sum, and we will subtract α_{w_i} from the final sum. The new game is shown in Figure 5.

The reason that these new addends do not change the adversary's view of the game stems from the fact that the $((y_{v_i} - R_{v_i}) \bmod b, (z_{w_i} - \alpha_{w_i}) \bmod a)$ in Game B retain the same distribution as (y_{v_i}, z_{w_i}) had in game B'.

We now have that $P_{\mathrm{Fe}} = \Pr[D^B = 1]$. The probability is taken over the random q-vectors x, y, and z with coordinates in $[0, 2^n - 1]$.

We now consider one final game, Game C. This game is identical to B except that we output $ay_i + z_i$ (instead of $a\beta_i + \gamma_i$). Obviously $P_{\mathrm{Rn}} = \Pr[D^C = 1]$. Again the probability is over the random vectors x, y, z.

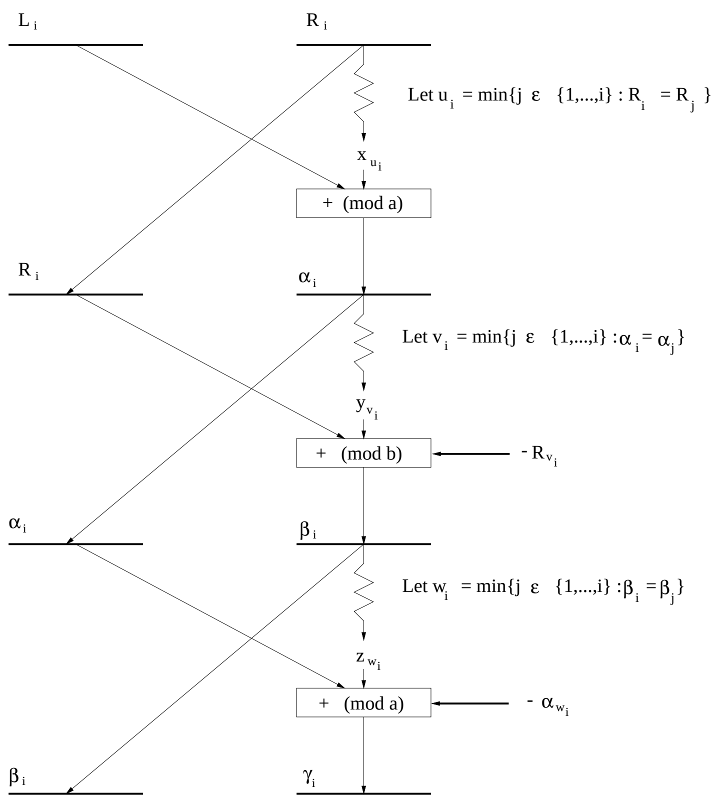

Fig. 5. *Game* B*. We modify B' by adding the quantities indicated by the emboldened arrows. This game is once again identical, from the adversary's perspective, to Game Fe.*

We will now make some observations and calculations about Games B and C which will allow us to conclude with the theorem. The idea is that Games B and C usually coincide. We will manage to bound adversarial advantage by looking at the chance that games B and C do not coincide.

First we define some events. These events are defined in Game C. (It is important that we do this in Game C, not Game B.) Define the event REPEAT_α as true if $\alpha_i = \alpha_j$ for some $i < j$–that is, some α_i arises twice. Define the event REPEAT_β as true if $\beta_i = \beta_j$ for some $i < j$–that is, some β_i arises twice. Define the event REPEAT as the disjunct of α_i and β_i–that is, either an α_i repeats or a β_i repeats. Again, these events are defined in Game C.

Claim. $\Pr[\mathsf{REPEAT}_\alpha] \leq \dfrac{q^2\lceil 2^n/a\rceil}{2^{n+1}}$.

Look at query i. If R_i itself is a repetition of an earlier R_j, then we know for sure that $\alpha_i \neq \alpha_j$, since all queries are assumed to be distinct. It is possible, however, that α_i could coincide with some α_j where R_j was different from R_i. But we have provided the adversary no information about internal x_i and α_i values. If the cardinality of $[0, 2^n - 1]$ were evenly divisible by a then we would know the chance for any particular α_j to coincide with α_i would be $1/a$. This is because we are taking the sum of L_i with a random member of $[0, 2^n - 1]$ and then taking this $\pmod a$. But of course 2^n may not be divisible by a and this modulus will create an "tail effect" slightly biasing the probability. We can easily measure this, however, as follows: the amount of probability mass on some points will be $\lfloor 2^n/a \rfloor / 2^n$ and on the others it will be $\lceil 2^n/a \rceil / 2^n$. We will simply take the latter as a bound. If R_i is a new, unrepeated value, then x_{u_i} will be a random number in $[0, 2^n - 1]$ and so the chance that α_i will collide with any particular prior α_j is again bounded by $\lceil 2^n/a \rceil / 2^n$. Thus the chance that α_i will collide with an earlier query is at most $(i-1)\lceil 2^n/a \rceil / 2^n$, and the chance that there will eventually be a collision in α_i-values is at most $\sum_{i=1}^{q}(i-1)\lceil 2^n/a \rceil / 2^n \le \frac{q^2}{2}\lceil 2^n/a \rceil / 2^n$. $\diamondsuit$

Claim. $\Pr[\mathsf{REPEAT}_\beta | \overline{\mathsf{REPEAT}_\alpha}] \le \dfrac{q^2\lceil 2^n/b \rceil}{2^{n+1}}$.

By assumption, the v_i values are all distinct, so y is being evaluated on distinct points. The chance that two β_i values coincide is determined similar to the case in the previous claim where the R_i values were distinct. So analogously we have $\sum_{i=1}^{q}(i-1)\lceil 2^n/b \rceil / 2^n \le \frac{q^2}{2}\lceil 2^n/b \rceil / 2^n$. $\diamondsuit$

Putting this together we have that

Claim. $\Pr[\mathsf{REPEAT}] \le \dfrac{q^2}{2^{n+1}}\left(\lceil 2^n/a \rceil + \lceil 2^n/b \rceil\right)$.

The reason is that

$$\begin{aligned}
\Pr[\mathsf{REPEAT}] &= \Pr[\mathsf{REPEAT}_\alpha] + \Pr[\mathsf{REPEAT}_\beta \wedge \overline{\mathsf{REPEAT}_\alpha}] \\
&= \Pr[\mathsf{REPEAT}_\alpha] + \Pr[\mathsf{REPEAT}_\beta \mid \overline{\mathsf{REPEAT}_\alpha}] \cdot \Pr[\overline{\mathsf{REPEAT}_\alpha}] \\
&\le \Pr[\mathsf{REPEAT}_\alpha] + \Pr[\mathsf{REPEAT}_\beta \mid \overline{\mathsf{REPEAT}_\alpha}],
\end{aligned}$$

and we have just bounded each of the above addends. $\diamondsuit$

Now for the key observation:

Claim. $\Pr[D^B = 1 \mid \overline{\mathsf{REPEAT}}] = \Pr[D^C = 1 \mid \overline{\mathsf{REPEAT}}]$.

Both probabilities are over random choices of x, y, z. On the right-hand we output y_i, z_i in response to the ith query. On the left-hand side, assuming that REPEAT does not hold *in Game* C, once again we output y_i, z_i. This would be clear if we had said "assuming that REPEAT does not hold *in Game* B," and we defined this even in Game B in the obvious manner. But notice that as long as REPEAT does not hold in Game C, Game C and Game B behave identically, always returning y_i, z_i in response to query i. This is easily established by induction. $\diamondsuit$

We claim that, because of the last claim,

$$\begin{aligned} P_{\mathrm{Fe}} - P_{\mathrm{Rn}} &= \Pr[D^B = 1] - \Pr[D^C] = 1 \\ &\le \Pr[\mathsf{REPEAT}] \end{aligned}$$

Let A, B, C be arbitrary events and assume $\Pr[A \mid \overline{C}] = \Pr[B \mid \overline{C}]$. Now

$$\begin{aligned} \Pr[A] - \Pr[B] = &\Pr[A \mid \overline{C}] \Pr[\overline{C}] + \Pr[A \mid C] \Pr[C] \\ &- \Pr[B \mid \overline{C}] \Pr[\overline{C}] - \Pr[B \mid C] \Pr[C] \end{aligned}$$

and so $\Pr[A \mid \overline{C}] = \Pr[B \mid \overline{C}]$ tells us that first and third addends cancel. Now upperbound the second addend by dropping the $\Pr[A|C]$ (that is, upperbound this by 1) and drop the final addend (which is negative) entirely, thereby getting an upperbound of $\Pr[C]$, as desired.

We now address the case where we iterate the cipher. In other words, what happens when $ab - k > 0$? In this case we may invoke $\mathrm{fe}[3, a, b]_K(\cdot)$ multiple times per encipherment, and we must account for this in the bound. The crucial point in the proof affected by iterating is when we are calculating REPEAT_α. In the worst case, the first encipherment could cause us to compute $\mathrm{fe}[3, a, b](m)$ for *all* $m \in [k, ab-1]$. In this case up to $ab-k$ values of α_i may already have been computed. We therefore include these points in the computation of $\Pr[\mathsf{REPEAT}_\alpha]$. The new bound is therefore $\sum_{i=1}^{q+(ab-k)} (i-1)\lceil 2^n/a \rceil / 2^n \le \frac{(q+ab-k)^2}{2} \lceil 2^n/a \rceil / 2^n$ for $\Pr[\mathsf{REPEAT}_\alpha]$ and similarly for $\Pr[\mathsf{REPEAT}_\beta \mid \overline{\mathsf{REPEAT}_\alpha}]$. So the overall bound is now

$$\Pr[\mathsf{REPEAT}] \le \frac{(q + ab - k)^2}{2^{n+1}} \left(\lceil 2^n/a \rceil + \lceil 2^n/b \rceil \right).$$

And setting $\Delta = ab - k$ we obtain the bound of Theorem 2.