REBEL: Reconfigurable Block Encryption Logic

Mahadevan Gomathisankaran and Akhilesh Tyagi
Iowa State University
Electrical & Computer Engineering
Ames, IA - 50011
{gmdev, tyagi}@iastate.edu

September 1, 2006

Abstract

Existing block cipher function designs have tended to deploy the secret bits in a specific and limited way. We generalize the role of the secret as truth tables of Boolean gates in a carefully designed logic schema. Our claims are: these reconfigurable functions are pseudo one-way and pseudo random functions. Such a function family is proposed using reconfigurable gates. Based on this function family we create REBEL, REconfigurable Block Encryption Logic, which is an LR-Network. We prove cryptographic and cryptanalytic security for REBEL. From cryptographic perspective, this function is a pseudo-permutation. From cryptanalysis perspective, any observable attribute appears to be a random process.

1 Introduction

Conventional block ciphers ([12, 13]) derive their security from an embedded secret, more commonly referred to as a key. One of the inputs, key, in each round is a secret, whereas the round functions themselves are public. This is a deliberate design decision so that the algorithm can be published as a standard. The secret, however, is combined with the state in a limited way, as an xor, during a round. The xor based mixing of the cipher state and the secret leads to some vulnerabilities based on linear and differential cryptanalysis. The complexity of extracting the secret or its properties is proportional to the non-linearity, among many other attributes, of the round functions.

We propose a simple yet novel approach wherein the round functions themselves become the secret, while the function schema is a publicly published algorithm. The intuition is to use reconfigurable gates as round functions and define their configurations as the secret (or key). Hence the complexity of such a cryptographic function is derived from the fact that almost all of the round processing is driven by the secret (truth tables). In a traditional block cipher, the secret is combined with the state with an xor as one of the steps in the round. This xor step is susceptible to linear modeling of the secret and input/output relationship. When the secret is used as a Boolean gate truth table, it is inherently non-linear, especially when many levels of such gates are composed, and when each gate is selected to have large minterm degree.

The advantage of such a reconfigurable block encryption logic (REBEL) is that it is highly time-efficient when implemented in hardware. The throughput needs of cryptographic blocks when placed in-line into a processor pipeline in order to support secure execution environments [7, 14] can be of the order of 20-30 GB/second. Based on our experience with a reconfigurable hash function [9], we believe that 100+...
GB/second encryption/decryption bandwidth is easily achievable for REBEL implementations. We believe that the hardware implementation advantage is a direct consequence of not basing our framework on finite field theory, as is common for the existing block ciphers. Another added advantage for REBEL is that the key-space is much larger than the block length of the encryption function. This provides much higher security levels without having to increase the block length.

We describe the preliminaries needed for the paper in Section 2. This is followed by the introduction of the function family that forms the basis for REBEL in Section 3. Section 4 presents the REBEL schema and its analysis. Section 5 concludes the paper.

2 Preliminaries

2.1 Notations

- $I_n$ denotes the set of all $n$-bit strings, $\{0, 1\}^n$.
- For $x \in I_{2n}$, denote by $x|_n$ the first (left) $n$ bits of $x$ and by $x|_{2n}$ the last (right) $n$ bits of $x$.
- $G^b_n$ denotes the set of all $I_n \mapsto I_1$ balanced gates (functions).
- Let $x$ and $y$ be two bit strings of equal length, then $x \oplus y$ denotes their bit-by-bit exclusive-or.
- Let "$\circ$" denote function composition operator, i.e., $f = g \circ g$ implies, $f(x) = g(g(x))$.
- Let "$\bullet$" denote concatenation of binary digits, i.e., $p = x \bullet y$ implies, $p$ is $2N$ bit string with left $N$ bits containing $x$ and right $N$ bits containing $y$.
- Let $f^{(n)}$ represent the composition of function $f$ $n$-times i.e., for example $f^{(3)} = f \circ f \circ f$.
- Let a gate $g$ be defined over $n$ input bits/variables $\{x_1, x_2, \ldots, x_n\}$. A literal $l$ is defined as either an input variable $x_i$ or its complement $\overline{x_i}$. A minterm is defined as a product of $n$ literals wherein no variable $x_i$ occurs both in complemented and uncomplemented literal form.
- $\in_R$ represents the action of choosing uniformly at random an element from a set.

2.2 Properties of $G^b_n$

2.2.1 Cardinality:

$$|G^b_n| = \left(\frac{2^n}{2n-1}\right).$$

2.2.2 Closed over complement:

**Definition 2.1.** A set of gates $G$ is closed over complement if $\forall g \in G$ the complement $\overline{g}$ is also present in $G$.

Consider a gate $g \in G^b_n$. Then $g$ is balanced and has equal number of zeros and ones in its truth table. The truth table of its complement gate $\overline{g}$ is nothing but the complement of every truth table row of $g$, i.e., every zero of $g$ will become one and every one of $g$ become zero. Hence the number of zeros and ones in $\overline{g}$ is still the same as $g$. Hence $\overline{g}$ is also balanced and is present in the set $G^b_n$. Thus the set $G^b_n$ is closed over complement.
2.2.3 Symmetry over Input Variables:

Any gate $g$ can be represented as a boolean function of input variables. Let the set of $n$ input variables to the gate $g$ be $I(g) = \{x_1, x_2, \ldots, x_n\}$. The support set of $g$, $\text{Sup}(g) \subseteq I(g)$, which is the set of all input variables $g$’s value depends on, i.e., if $x_i \in \text{Sup}(g)$ then $\delta g/\delta x_i \neq 0$. A set of gates $G$ is symmetric if for each $g \in G$, if any pair of input variables in $\text{Sup}(g)$ are swapped in the expression for $g$ leading to a gate $g'$, then $g' \in G$. This property exhibits $G$’s bias towards certain input variables. For example if the set $G$ has more gates with variable $x_i$ than any other variables then the set becomes biased towards $x_i$ and any change in $x_i$ is reflected at the output with higher probability than a change in any other input variables.

**Definition 2.2.** A set of gates $G$ is said to be symmetric in input variables if $\forall g \in G$ the gate $g'$ obtained by swapping any two input variables from the support set of $g$ is also present in $G$.

For $g \in G_n^b$, the balance property dictates that there be $2^{n-1}$ minterms. A swap of $x_i$ and $x_j$ in the expression for $g$ does not alter the number of 1’s (minterms). It merely, recodes the minterm locations. Hence the gate $g'$ derived from $g$ through a swap of two variables will also have $2^{n-1}$ minterms, and hence will be balanced, implying $g' \in G_n^b$. Hence, the set $G_n^b$ is symmetric over input variables. Now on, we will use the term symmetric set $G$ to mean a set symmetric in input variables.

2.2.4 Closed over variable-complement:

**Definition 2.3.** We define a set of gates $G$ as closed over input variable-complement if $\forall g \in G$ the gate $g'$ obtained by complementing all literal instances of an input variable $x_i$ is also present in $G$.

Let $g \in G_n^b$ and let $g'$ be obtained by complementing the variable $x_i$. Since $g$ is balanced the number of 1’s (minterms) is $2^{n-1}$. Each affected minterm gets relocated to a different row in the truth table through complementing of a literal $x_i$ or $\overline{x_i}$. However, the number of minterms still remains at $2^{n-1}$, and hence $g'$ is also balanced. Thus $G_n^b$ is closed over input variable-complement.

2.2.5 Input-collision probability:

**Definition 2.4.** For any gate $g$, the input-collision probability $p_{\text{coll}}^g$ is defined as the probability that any pair of inputs $x, x' \in R_n$ s.t. $x \neq x'$, produce the same output, i.e., $p_{\text{coll}}^g = P[g(x) = g(x') \mid g]$. Note that the collision probability is averaged over all the input pairs.

For every $g \in G_n^b$ there are equal number $(2^{n-1})$ of 0’s and 1’s. For the collision to happen, the input $x$ has $2^n$ out of $2^n$ choices and the input $x'$ has $2^{n-1} - 1$ out of $2^n - 1$ choices. Hence $p_{\text{coll}}^g = \frac{2^{n-1}-1}{2^n-1}$.

2.2.6 Gate-collision probability:

**Definition 2.5.** For any given input $x \in I_n$ gate-collision probability is defined as the probability that any pair of gates $g, g' \in R$ collide i.e., $p_{\text{coll}}^G(x) = P[g(x) = g'(x) \mid x]$. Note that the collision probability is averaged over all the gate pairs.

The set $G_n^b$ is closed over complement. Hence for every $g \in G_n^b$ the complement gate $\overline{g}$ is also present in $G_n^b$. Thus every row of the truth table for the set is also balanced and $p_{\text{coll}}^{\overline{g}}(x) = \frac{1}{2}$.

2.2.7 Set-collision probability:

**Definition 2.6.** For any set of gates $G$, the set-collision probability $p_{\text{coll}}^G(x, x')$ is defined as the probability that any gate $g \in R$ $G$ collides for any given pair of inputs $x, x' \in I_n$ s.t. $x \neq x'$, i.e., $p_{\text{coll}}^G(x, x') = \ldots$
\[ P[G(x) = G(x') \mid x, x'], \text{ where } G(x) = \frac{\sum_{g \in G} g(x)}{|G|}. \] Note that the collision probability is averaged over all the gates in the set \( G \).

In \( G_n^b \) for every input pair \( x, x' \) the number of gates that collide are \( N_{\text{coll}} = 2 \times \left( \frac{2^n - 2}{2^{n-1} - 2} \right) \) and the number of gates that does not collide are \( N_{\text{coll}} = 2 \times \left( \frac{2^n - 2}{2^{n-1} - 1} \right) \). Hence the probability of collision is

\[ P_{\text{coll}}^G = \frac{N_{\text{coll}}}{N_{\text{coll}} + N_{\text{coll}}} = \frac{2^{n-1} - 1}{2^n - 1}. \]

In other words, the set \( G_n^b \) is symmetric with respect to every pair \( x, x' \).

### 2.2.8 Input symmetric-controllability:

**Definition 2.7** (input pairs equivalence class). Let \( p, p' \in I_n \) be the inputs to the gates in the set \( G \). We group the pairs of inputs based on their hamming distance, i.e., we define equivalence class as \( S_d = \{(p, p') \mid h(p, p') = d\} \).

**Definition 2.8.** A set of gates \( G \) is input symmetric-controllable if the collision probability is same for any pair of inputs in the same equivalence class.

**Lemma 2.1.** Any symmetric set of gates \( G \) and closed over variable complement is input symmetric-controllable.

**Proof.** Let \( (p, p'), (q, q') \in S_d \) then the input-symmetry property requires that \( P[G(p) = G(p')] = P[G(q) = G(q')] \). The trivial cases are \( p = q, p' = q' \) and \( p = q', p' = q \). We will prove the non-trivial cases. Let \( i_1, i_2, \ldots, i_d \) s.t. \( i_1 < i_2 < \ldots < i_d \) represent the \( d \) bit-positions in which the pair \((p, p')\) differ and similarly \( j_1, j_2, \ldots, j_d \) s.t. \( j_1 < j_2 < \ldots < j_d \) represent the \( d \) bit-positions in which the pair \((q, q')\) differ.

**Case** \( \{i_1, i_2, i_3, \ldots, i_d\} = \{j_1, j_2, j_3, \ldots, j_d\} \):

Let \( k_1, k_2, \ldots, k_{n-d} \) be the bit-positions which do not differ in both the pairs. Then \( p \) and \( q \) should differ in at least one of these bit-positions. Let \( g \in G \) be the gate that collides for the pair \((p, p')\). Let \( g' \) be a gate obtained by changing the literals of \( g \) in the following way. \( x_{k_1}^{g'} = x_{k_1}^{q} \oplus p_{k_1} \oplus q_{k_1} \), that is, the literals are complemented in bit-positions where \( p \) and \( q \) differ. But \( G \) is close over literal-complement. Hence \( g' \) is also present in the set \( G \) and \( g' \) will collide for the pair \((q, q')\).

**Case** \( \{i_1, i_2, i_3, \ldots, i_d\} \neq \{j_1, j_2, j_3, \ldots, j_d\} \):

Let \( g \in G \) be a gate that collides for the pair \((p, p')\). Let \( g' \) be a gate such that the input variables are swapped in the following way. Variables \( x_{i_k}^{g'} = x_{j_k}^{q} \), \( x_{i_j}^{g'} = x_{j_k}^{q} \), that is the variables at positions \( i_k \) and \( j_k \) are swapped. But the set \( G \) is symmetric. Hence \( g' \) is also present in the set \( G \) and \( g' \) will collide for the pair \((y, y')\).

Thus for every gate \( g \) that collides for the pair \((p, p')\), a gate \( g' \) that collides for the pair \((q, q')\), exists. Hence the probability of collisions are same or \( P[G(p) = G(p')] = P[G(q) = G(q')] \).

**Corollary 2.1.** \( G_n^b \) is input symmetric-controllable.

**Proof.** \( G_n^b \) is symmetric and closed over variable-complement (Properties 2.2.3 and 2.2.4 ).
2.2.9 \textit{k-wise set-collision probability}:

Let $g_i$ represent $i^{th}$ gate that is chosen independently at random from $G_n^b$. Let $k_p^{G_{\text{coll}}} = \bigcap_{i=1}^{k} P[g_i(x) = g_i(x')]$ be the probability that all the $k$ gates collide for any given pair of inputs $x, x' \in I_n$ s.t. $x \neq x'$. Then $k_p^{G_{\text{coll}}} = (p^{G_{\text{coll}}})^k$.

From Property 2.2.7 for every pair of inputs $x$ and $x'$ number of gates that collide in the set $G_n^b$ is $N_{\text{coll}}$. That is, the probability of picking a gate from $G_n^b$ such that it collides for any input pair $x$ and $x'$ is $N_{\text{coll}}|G_n^b|$. And this probability is same for all the choices of the gates since they are chosen independently and at random. Hence $k_p^{G_{\text{coll}}} = (p^{G_{\text{coll}}})^k$.

2.2.10 \textit{bias propagation}:

Let $x \in I_n$ be the input to gate $g \in G_n^b$. Let $x_i$ be the $i^{th}$ bit. Let $\epsilon_{\text{in}}$ be the bias of the input bits i.e., the probability that the input bit $i$ collides is $\frac{1}{2} + \epsilon_{\text{in}}$. Then the bias at the output is

$$
\epsilon_{\text{out}} = \left(\frac{1}{2} + \epsilon_{\text{in}}\right)^n + \left(1 - \left(\frac{1}{2} + \epsilon_{\text{in}}\right)^n\right) \cdot \frac{2^n - 1}{2^n - 1} - \frac{1}{2}
$$

(1)

And,

$$
|\epsilon_{\text{out}}| \leq |\epsilon_{\text{in}}|
$$

(2)

3 \textbf{Function Family} $F_n^N$

**Definition 3.1 (F_n^N function family).** $F_n^N$ is a $I_N \mapsto I_N$ family of functions that uses $I_n \mapsto I_1$ reconfigurable gates as the component gates, where, $N = n^{\lceil \log_n N \rceil}$.

3.1 \textbf{Notations}

- Let $\gamma$ be a reconfigurable gate.
- Let $x(\gamma)$ and $y(\gamma)$ be the input and output of the reconfigurable gate respectively.
- Let $g(\gamma)$ be the gate implemented by (or the truth table of) the reconfigurable gate.
- Let $\Gamma$ be a $I_N \mapsto I_1$ tree using $I_n \mapsto I_1$ reconfigurable gates (or $n$-ary tree). Then there are $\log_n N$ levels in $\Gamma$ (wlog let top level be 1) and $i^{th}$ level has $N_i$ reconfigurable gates (wlog let leftmost gate be 1). Figure 1 shows the diagrammatic representation of this construction.
- Let $N_{\Gamma}^i$ be the number of gates present at $i^{th}$ level.
- Let $\gamma_{i,j}(\Gamma)$ be the reconfigurable gate at $i^{th}$ level $j^{th}$ position.
- Let $\downarrow$ represent switching of any bit $i.e., P[b \downarrow]$ represent the probability that the bit $b$ switches.
Figure 1: Diagramatic representation of $\Gamma$

Figure 2: Diagrammatic representation of $F^N_n$
3.2 Construction of $F^N_n$

$F^N_n$ is a $I_N \mapsto I_N$ family of functions that uses $I_n \mapsto I_1$ reconfigurable gates as the component gates. $F^N_n$ has $N$ tree-gates ($\Gamma$) each one producing one output bit using the same input bits. Let $\Gamma_i$ be the $i^{th}$ tree-gate (wlog let leftmost be 1). Figure 2 shows this construction. In the figure, $x$ represents the $N$ bit input and $y_i$ represent the $i^{th}$ bit of $y$ which is the $N$ bit output.

The configurations of the reconfigurable gates are connected as $g(\gamma_{i,j}(\Gamma_k)) = g(\gamma_{p,q}(\Gamma_r))$ where,

$$i = 1 + (\sum_{t=1}^{i-1} N_t^\Gamma + j + k - 2) \% N = (\sum_{t=1}^{p-1} N_t^\Gamma + q + r - 2) \% N$$

and $1 \leq i, p \leq \log N, 1 \leq j \leq \frac{N}{n^i}, 1 \leq q \leq \frac{N}{np},$ and $1 \leq k, r \leq N$. Thus only $N$ unique gate configurations are required by $F^N_n$. The configurations are chosen uniformly at random from the set $G^N_n$. The configurations forms the key to function family.

Thus the key space is $\mathcal{K} = \{g \mid g \in G^N_n\}$ and its cardinality is $|\mathcal{K}| = (|G^N_n|)^N$. Let $\kappa \in \mathcal{K}$ and $\kappa_i$ be the $i^{th}$ gate configuration. Then the gate configuration assignments are done as, $g(\gamma_{p,q}(\Gamma_r)) = \kappa_i$ where

$$i = 1 + (\sum_{t=1}^{p-1} N_t^\Gamma + q + r - 2) \% N \text{ and } 1 \leq p \leq \log N, 1 \leq q \leq \frac{N}{np}, \text{ and } 1 \leq i, r \leq N.$$

The construction can be explained in simpler terms as follows. Consider $N$ columns and $\frac{N-1}{n-1}$ rows of reconfigurable gates. Then $i^{th}$ column represents the gates of the tree-gate $\Gamma_i$ and $j^{th}$ row in a column corresponds to the $q^{th}$ gate in $p^{th}$ level such that $j = \sum_{t=1}^{p-1} N_t^\Gamma + q$. In every row there are $N$ reconfigurable gates.
gates and are assigned \( N \) gates from the key \( i.e., \gamma_{i,j} \) represent the reconfigurable gate in \( i^{th} \) column and \( j^{th} \) row, then \( g(\gamma_{i,j}) = \kappa_{(i+j-2)\%N+1} \). Figure 3 shows this construction diagrammatically.

### 3.3 Properties of \( F^n_N \)

**Definition 3.2** (input group). \( N \) bits of input \( x \) to \( F^n_N \) are divided into equal sized groups of size \( n \) bits. Bits of group \( -i \) of \( x \), denoted by \( grp(x,i) \), correspond to the input bits of \( i^{th} \) reconfigurable gate in level-1 (top level), bits \( x_{(i-1)n+1}, x_{(i-1)n+2}, \ldots, x_{in} \).

**Definition 3.3** (\( \Delta_g \)-difference operator). \( \Delta_g \) is a difference operator that takes two \( N \) bit values and returns the number of groups in which the two inputs differ, \( i.e., \Delta_g(x,x') = i \), where \( i \in \{0, N/n\} \) and \( x, x' \in I_N \). Let \( \delta_g(x,x') = \{i_1, i_2, \ldots, i_k\} \) such that \( 0 \leq k \leq \Delta_g(x,x') \) be the set of group indices where \( x \) and \( x' \) differ.

**Definition 3.4** (group hamming distance operator). Given two \( N \) bit values \( x, x' \in I_N \) the group hamming distance operator \( h_{\Delta_g} \) is defined as, \( h_{\Delta_g}(x,x') = [h_{i_1}, h_{i_2}, \ldots h_{i_{N/n}}] \), the set of groupwise Hamming distances in ascending order, \( i.e., h_{im} = h(grp(x,i_m), grp(x',i_m)) \) and \( h_{ip} \leq h_{iq} \) for \( p < q \).

#### 3.3.1 bit-Balance:

Every output bit of every function \( f \in F^n_N \) is balanced. This property is derived from the fact that every output is produced by a \( n \)-ary tree of balanced gates. Let \( x, y \in I_N, f \in F^n_N, y = f(x) \) and \( y_i \) be the \( i^{th} \) output bit. When averaged over all the inputs to the tree-gate \( \Gamma_i \), \( P[\Gamma_i(x) = 0] = P[\Gamma_i(x) = 1] \), hence \( P[y_i = 1] = P[y_i = 0] \).

#### 3.3.2 key-collision probability:

**Definition 3.5.** For the function family \( F^n_N \) key-collision probability \( p_{kcol} \) is defined as the probability that for a given input \( x \in I_N \) any pair of keys \( \kappa, \kappa' \in R \) produce the same output \( i.e., p_{kcol}(x) = P[f_{\kappa'}(x) = f_{\kappa'}(x) | x] \). Note that the probability is averaged over all the pairs of keys \( \kappa, \kappa' \).

To derive this property let us consider each tree individually. Let \( \Gamma_i^\kappa \) and \( \Gamma_i^{\kappa'} \) be the tree \( i \) with keys \( \kappa \) and \( \kappa' \) respectively. Then the output of trees collide if \( \gamma_{\log_n N,1}(\Gamma_i^\kappa) = \gamma_{\log_n N,1}(\Gamma_i^{\kappa'}) \). Even though inputs to the trees are the same the inputs to the root node are independent. This follows from the gate-collision property (Property 2.2.6) of \( G^n_N \). Consider two level 1 gates \( \gamma_{j,1}(\Gamma_i^\kappa) \) and \( \gamma_{j,1}(\Gamma_i^{\kappa'}) \). Let \( b_j \) and \( b_j' \) be their outputs respectively. Then, \( P[b_j = x \land b_j' = y] = P[b_j = x] \cdot P[b_j = y] \) where \( x, y \in \{0, 1\} \). This is because,

\[
\begin{align*}
P[b_j = 0 \land b_j' = 0] &= P[b_j \text{ and } b_j' \text{ collide}] \quad \cdot \quad P[b_j = 0] = \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4} \\
P[b_j = 0 \land b_j' = 1] &= P[b_j \text{ and } b_j' \text{ not collide}] \quad \cdot \quad P[b_j = 0] = \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4} \\
P[b_j = 1 \land b_j' = 0] &= P[b_j \text{ and } b_j' \text{ not collide}] \quad \cdot \quad P[b_j = 1] = \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4} \\
P[b_j = 1 \land b_j' = 1] &= P[b_j \text{ and } b_j' \text{ collide}] \quad \cdot \quad P[b_j = 1] = \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}
\end{align*}
\]

Thus output bits of every level 1 gates \( \gamma_{j,1}(\Gamma_i^\kappa) \) and \( \gamma_{j,1}(\Gamma_i^{\kappa'}) \) are independent. Hence the input bits to the root node are independent. Let \( g_\gamma = g(\gamma_{\log_n N,1}(\Gamma_i^\kappa)) \) and \( g_\gamma' = g(\gamma_{\log_n N,1}(\Gamma_i^{\kappa'})) \) and let \( x_\gamma \) and \( x'_\gamma \) be the inputs to these gates respectively. Then, probability of key-collision for a tree is given by \( P[g_\gamma(x_\gamma) = g_\gamma'(x'_\gamma)] = g_\gamma'(x'_\gamma) \). Since both the gates \( g_\gamma \) and \( g_\gamma' \) are balanced the probability of collision is nothing but \( \frac{1}{2} \). Since root nodes of every tree are independent, for any input \( x \), \( p_{kcol}(x) = \frac{1}{2^N} \).
Thus for any input \( x \in I_N \)

\[
p_{\text{kecol}}(x) = \frac{1}{2^N} \tag{3}\]

Another property which follows the key-collision property is the probability that \( k \) inputs \( x_1, x_2, \ldots, x_k \) chosen uniformly at random from \( I_N \) collide i.e., \( f_\kappa(x_i) = f_{\kappa'}(x_i) \) \( \forall 1 \leq i \leq k \). Since every input is independent this is nothing but \( \frac{1}{2^{Nk}} \).

### 3.3.3 Tree independence:

For any function \( f \in R F^N_n \) the collision probability for any given pair of inputs, \( x, x' \in I_N \) s.t. \( x \neq x' \) is \( P[f(x) = f(x') \mid (x, x')] = \prod_{i=1}^{N} P[\Gamma_i(x) = \Gamma_i(x') \mid (x, x')] \). Wlog let \( x \) and \( x' \) differ in \( k \) groups denoted by \( i_1, i_2, \ldots, i_k \). From the \( k \)-wise set-collision property of \( G^b_n \) (Property 2.2.9) the collision probabilities of \( N \) level-1 gates in each of the groups \( i_1, i_2, \ldots, i_k \) are independent. Similarly the collision probabilities of \( N \) gates in the same group in subsequent levels are also independent. Hence the collision probabilities of every tree becomes independent. Thus,

\[
P[f(x) = f(x') \mid (x, x')] = \prod_{i=1}^{N} P[\Gamma_i(x) = \Gamma_i(x') \mid (x, x')] \]

Since each tree is similar in structure, the collision probabilities of every tree is same. Thus,

\[
P[f(x) = f(x') \mid (x, x')] = \left( P[\Gamma(x) = \Gamma(x') \mid (x, x')] \right)^N \]

### 3.3.4 Collision controllability:

**Definition 3.6.** Collision controllability \( \delta_c \) is the maximum probability with which any two inputs \( x, x' \in I_N \) to the function \( f \in R F^N_n \) cause collision, i.e., \( \delta_c = \max_{x, x'} \{ P[f(x) = f(x')] \} \).

From Property 3.3.3 the collision probabilities of trees are independent and equal. Hence to maximize the collision probability of \( f \), the collision probability of tree has to maximized. The collision probability of the tree gate will be maximum if inputs to only one gate on the top level switch, i.e., requiring minimum number of gates in the tree to collide. Figure 4 shows one such path in \( \Gamma \). Thus,

\[
\max_{x, x'} \{ P[\Gamma_i(x) = \Gamma_i(x')] \} = 1 - (1 - p_{\text{gcol}}^g)^{\log_n N} = 1 - \left( \frac{2^{n-1}}{2^n - 1} \right)^{\log_n N}
\]

\[
\Rightarrow \delta_c = \left( 1 - \left( \frac{2^{n-1}}{2^n - 1} \right)^{\log_n N} \right)^N \tag{4}
\]

### 3.3.5 Function family collision probability:

**Definition 3.7.** Function family collision probability is the probability that any two inputs \( x, x' \in R I_N \) s.t. \( x \neq x' \) cause collision in any function \( f \in R F^N_n \) i.e., \( p_{\text{fcol}}^f = P[f(x) = f(x')] \). Note that the probability is averaged over all the input pairs and all the functions in the family.
Figure 4: One of the maximum controllability paths in \( \Gamma \)
Let $i = \Delta_g(x, x')$ and $p^i_{\text{group}}$ be the probability that the inputs with $\Delta_g = i$ cause collision. Let $N^i_{\text{group}}$ be number of inputs with $\Delta_g = i$. Then,

$$P^f_{\text{coll}} = \frac{\sum_{i=1}^{N} N^i_{\text{group}} p^i_{\text{group}}}{2^N (2^N - 1)}$$

(5)

### 3.3.6 Input-symmetry:

The function family $F^N_n$ exhibits the following input symmetry. Let $x, x', y, y' \in I_N$, $f \in F^N_n$ and $y = f(x), y' = f(x')$. Let $y_i, y'_i$ be the $i^{\text{th}}$ output bit of $y$ and $y'$ respectively. Then,

$$P[y_i = y'_i | (x, x')] = P[y_j = y'_j | (x, x')]$$

That is, the probability of collision for every output bit is same for any pair of inputs when averaged over all the functions in the family $F^N_n$.

**Definition 3.8.** We define the property input-symmetry as the probability of collision of every output bit being equal for any input change.

The trivial case is when $x = x'$. We will see the non-trivial case when $x \neq x'$. The collision probability of an output bit is nothing but the collision probability of the corresponding tree $\Gamma_1$. From Property 3.3.3 the collision probabilities of trees are independent and equal for a given pair of inputs. Hence, $P[y_i = y'_i | (x, x')] = P[y_j = y'_j | (x, x')]$ for any pair of inputs $x, x'$.

### 3.3.7 Output-symmetry:

The function family $F^N_n$ exhibits the following output symmetry. For a randomly selected instance of $f$ from $F^N_n$, the input transition that causes an output transition from $y$ to $y'$ is indistinguishable if the inputs come from the same group equivalence class. This prevents an adversary from extracting any information about the output values of instances that cause the observed output change, which captures a typical attack scenario where only the output is observable. We will describe an attack method in Section 3.3.10 where through carefully controlled differential input pairs, it may be possible to infer almost all the top level gates. Output symmetry prevents this occurrence. The preceding property makes it hard to distinguish between two vectors that cause a $y \rightarrow y'$ change. Moreover, even a model that attempts to ascertain a specific input bit position transition, $x_i \rightarrow \overline{x}_i$ and $x_j \rightarrow \overline{x}_j$, is also indistinguishable since all the input bit positions cause the output $y \rightarrow y'$ change uniformly. Quantitatively, let $f \in_R F^N_n$. 

$$P[x_i \downarrow | (y, y')] = P[x_j \downarrow | (y, y')]$$

That is, the probability of switching for every input bit position is same for any pair of observed output when averaged over all the functions in the family $F^N_n$.

**Definition 3.9.** A function family $F$ is output-symmetric if for a randomly chosen function instance $f \in_R F$, for a given output pair $(y, y')$, $P[x_i \downarrow | (y, y')] = P[x_j \downarrow | (y, y')]$ for all $1 \leq i, j \leq N$.

**Definition 3.10.** Two pairs of inputs $(p, p')$ and $(q, q')$ are in the same group-equivalence class iff the following condition holds. $h_{\Delta_g}(p, p') = h_{\Delta_g}(q, q')$ which also implies $\Delta_g(p, p') = \Delta_g(q, q')$. 

11
Let \( f \in_R F_n^N \), \( x, y \in I_N \) such that \( y = f(x) \). Let \( x', x'' \in_R I_N \) such that the pairs \((x, x')\) and \((x, x'')\) are in the same group-equivalence class. Then given that \( y \) has changed to \( y' \) output-symmetry property of \( F_n^N \) requires that both these pairs be equally probable to have caused the transition. Let \( P^f[y, y'] \mid (x, x') \) be the probability that \( y \) transitions to \( y' \) given that the input transitions from \( x \) to \( x' \). Then from tree independence property and input-symmetry property (Properties 3.3.3 and 3.3.6),

\[
P^f[y, y'] \mid (x, x') = (P[\Gamma(x) = \Gamma(x') \mid (x, x')])^{N-h(y,y')} (1 - P[\Gamma(x) = \Gamma(x') \mid (x, x')])^{h(y,y')}
\]

Let \( h_i \) and \( h_i'' \) be the hamming distance of the group-\( i \) of the pairs \((x, x')\) and \((x, x'')\) respectively. Then the sequence \( h_1', h_2', \ldots, h_{N/n}' \) is a permutation of the sequence \( h_1'', h_2'', \ldots, h_{N/n}''' \). The collision probabilities of the level-1 (top level) gates are independent of their group numbers as the gates are chosen uniformly at random for every group position. Hence the collision probabilities of the tree gates for the pairs \((x, x')\) and \((x, x'')\) are equal. Thus,

\[
P^f[y, y'] \mid (x, x') = P^f[y, y'] \mid (x, x'')
\]

But,

\[
P^f[(x, x') \mid (y, y')] = \frac{P^f[(y, y') \mid (x, x')] P[(x, x')]}{P^f[y, y']}
\]

Using the result of Equation 6,

\[
P^f[(x, x') \mid (y, y')] = \frac{P^f[(y, y') \mid (x, x'')] P[(x, x')]}{P^f[y, y']}
\]

But, \( P[(x, x')] = P[(x, x'')] \) as both are in the same group-equivalence class. Thus,

\[
P^f[(x, x') \mid (y, y')] = \frac{P^f[(y, y') \mid (x, x'')] P[(x, x'')]}{P^f[y, y']}
\]

\[
P^f[(x, x') \mid (y, y')] = P^f[(x, x'') \mid (y, y')]
\]

The construction of group-equivalence class ensures that every group is equally probable and every bit within a group is equally probable or in other words the number of transitions for every bit within a group-equivalence class is same. Hence from Equation 7,

\[
P^f[x_i = x'_i \mid (y, y')] = P^f[x_j = x'_j \mid (y, y')]
\]

### 3.3.8 Neighborhood Probability:

**Definition 3.11 (Key hamming distance).** We define the hamming distance of key \( \kappa \) and \( \kappa' \) as the number of gate configurations they differ. The operator \( \Delta_k \) returns hamming distance of keys \( \kappa \) and \( \kappa' \).

In order to use the function family \( F_n^N \) in cryptographic functions neighborhood property needs to be studied.

**Definition 3.12 (**\( \{d_x, d_y, d_k\}\)-neighborhood probability**). Let \( x, y, \kappa \) and \( x', y', \kappa' \) be any two sets s.t \( y = f_\kappa(x) \) and \( y' = f_{\kappa'}(x') \). Let \( h() \) be the bit-hamming distance function. Then their neighborhood probability is defined as \( P_{\text{coll}}^{\{d_x, d_y, d_k\}} = P[h(y, y') = d_y \mid \Delta_k(\kappa, \kappa') = d_k \cap h(x, x') = d_x] \). Note that this probability is averaged over all the inputs and all possible key pairs.
Deriving a generic expression for \( \{d_x, d_y, d_k\} \)-neighborhood probability is very cumbersome. Hence we will derive only for \( d_k = 1 \). Consider the case when \( d_x = 0 \) and \( d_y = 0 \). Consider a tree \( \Gamma \) in which only a level-1 gate is changed. Then,
\[
P[\text{collision in } \Gamma] = 1 - P[\text{no collision in } \Gamma] \\
= 1 - P[\text{no collision in top level } \cap \text{ no collision in subsequent levels}] \\
= 1 - P[\text{no collision in top level}] \cdot P[\text{no collision in subsequent levels}] \\
= 1 - \left( \frac{1}{2} \cdot \left( 1 - \frac{2^{n-1} - 1}{2^n - 1} \right) \log_2 N - 1 \right) \\
p_{coll}^{\{0,0,1\}}(\Gamma) = 1 - \left( \frac{1}{2} \cdot \left( \frac{2^{n-1} - 1}{2^n - 1} \right) \log_2 N - 1 \right) \\
< 1 - \left( \frac{1}{2} \right)^{\log_2 N}
\]
But every configuration in the key is reused \( \frac{N-1}{n-1} \) times. Hence,
\[
p_{coll}^{\{0,0,1\}} < P[\text{collision in } \Gamma] \left( \frac{N-1}{n-1} \right) \\
< \left( p_{coll}^{\{0,0,1\}}(\Gamma) \right) \left( \frac{N-1}{n-1} \right) \\
< \left( 1 - \left( \frac{1}{2} \right)^{\log_2 N} \right) \left( \frac{N-1}{n-1} \right)
\]
Note that in the above expression the inequality arises due to the following fact. Not all the \( \frac{N-1}{n-1} \) copies of the same configuration will be present in the top level and as the level where the gate is present increases the collision probability decreases. Hence the collision probability calculated by keeping all the gates in the top level becomes upper bound.

Similarly,
\[
p_{coll}^{\{0,d_y,1\}} < \left( \frac{N-1}{n-1} d_y \right) \cdot \left( p_{coll}^{\{0,0,1\}}(\Gamma) \right) \left( \frac{N-1}{n-1} - d_y \right) \cdot \frac{1}{2^{d_y}} \\
p_{coll}^{\{0,d_y,1\}} < \left( \frac{N-1}{n-1} d_y \right) \cdot \left( 1 - \left( \frac{1}{2} \right)^{\log_2 N} \right) \left( \frac{N-1}{n-1} - d_y \right) \cdot \frac{1}{2^{d_y}} \quad (9)
\]
The neighborhood probability for an input change of \( d_x \) and key hamming distance of 1 can be estimated as follows. From the Property 3.3.4 the maximum collision controllability is \( \left( 1 - \left( \frac{2^{n-1} - 1}{2^n - 1} \right) \log_2 N \right)^N \). Since the key has a hamming distance of 1, \( \frac{N-1}{n-1} \) configurations would have changed from \( f_n \) to \( f_{n'} \). When the configurations are different the collision probability increase from \( \frac{2^{n-1} - 1}{2^n - 1} \) to \( \frac{1}{2} \). Hence by taking the upper bound of the collision probabilities of the trees (\( \Gamma \)) which are affected by the configuration change we can find upper bound to \( p_{coll}^{\{d_x,0,1\}} \). Hence,
\[
p_{coll}^{\{d_x,0,1\}} < \left( 1 - \left( \frac{2^{n-1} - 1}{2^n - 1} \right) \log_2 N \right)^N \left( \frac{N-1}{n-1} \right) \left( \frac{1}{2} \right)^{\log_2 N} \\
p_{coll}^{\{d_x,0,1\}} < \left( 1 - \left( \frac{1}{2} \right)^{\log_2 N} \right)^N \left( \frac{N-1}{n-1} \right) \left( \frac{1}{2} \right)^{\log_2 N} \quad (10)
\]
The other useful inequality is
\[
p_{coll}^{\{d_x,d_y,1\}} \leq p_{coll}^{\{d_x,0,1\}} \quad (11)
\]
This is because the collision probability of every tree is always greater than \( \frac{1}{2} \) and forms the upper bound.
3.3.9 bias propagation:

From the Property 2.2.10 of the set $G^b_n$ the bias at the output of a gate $g \in G^b_n$ is less than the input bias. Thus the bias reduces as it propagates through the gates. In $F^N_n$ the bias of every tree $\Gamma$ is independent. In a tree there are $\log_n N$ levels of gates hence the output bias is lesser than the input bias.

3.3.10 Observability weakness:

One of the main design goals of $F^N_n$ is to make every output bit as function of all the input bits. The second design goal is to minimize the number of configurations. These two design goals itself forms the weakness of the function family. Consider a black box implementation of $f_n \in F^N_n$. In order to identify the secret (configurations) the approach would be to generate a pair of inputs $x, x'$ such that they differ in only one group (say $i$) i.e., $\Delta_i(x, x') = 1$. Let $y = f_n(x), y' = f_n(x')$ be the outputs and $r = grp(x, i)$ and $r' = grp(x', i)$. Let $b = y \oplus y'$ be the difference in the outputs and $b_j$ represent $j$th bit of $b$. Then if $b_j = 1$ then the configuration in $\gamma_{1,j}(\Gamma_i) = \kappa_j$ differs in rows $r$ and $r'$, i.e., $\kappa_k[r] \neq \kappa_k[r']$. The expected number of 1’s in $b$ is nothing but $N \cdot (1 - p^g_{coll})\log_n N$. Hence to find out about the characteristic of all the $N$ configurations for the rows $r$ and $r'$ a minimum of $\frac{1}{(1 - p^g_{coll})\log_n N}$ is needed. Hence by doing this experiment for all $2^{n-1}$ unique pairs of $r$ and $r'$ it is possible to find out which rows are similar and which rows are different. Once the relationship between the rows are identified the key space reduces to $2^{2N}$ bits. This is much smaller compared to the actual key space $|G^b_n|^N$.

3.3.11 Strength of $f^{(2)}$, $f \in F^N_n$

Observability: The weakness described in Section 3.3.10 arises due to the fact that the inputs to the top level gates ($\gamma_{1,j}(\Gamma)$) are controllable and the transitions (or switching) at the output of top level gates ($y(\gamma_{1,j}(\Gamma))$) are observable at the output of the tree ($y(\Gamma)$). Thus the controllability and observability should be de-linked in order to improve the strength of the function. Precisely this is achieved by the function composition $f^{(2)}$.

In $f^{(2)}$, let $x$ be the input, $y$ be the output, and $z$ be the output at midpoint i.e., $y = f^{(2)}(x)$ and $z = f(x)$. The observable output $y$ does not provide any information about the switching of any particular intermediate bit $z_i$ since all of them are equally probable to switch as shown in output-symmetry property (Property 3.3.7). The function family $F^N_n$ also is input-symmetric (Property 3.3.6) hence the adversary can not control the intermediate bits in a biased manner to observe its effect at the output. Thus the method used to extract information about top level gates truth table rows as explained in Section 3.3.10 is not applicable to $f^{(2)}$. We believe that given polynomial input output pairs of $f^{(2)}$ deciphering $f$ is an NP-hard problem. But we do not have a proof for this property at this time.

Neighborhood probability dispersion: The composition $f^{(2)}$ or in general $f^{(n)}$ decreases (or disperses) the neighborhood collision probability. In effect the composition makes the distribution more uniform and random. Let $y = f^{(2)}_n(x) = f_n(z)$ where $z = f_n(x)$. Then

$$p^{(0,d_g,1)}_{coll}(f^{(2)}) = \sum_{i=0}^{N-1} P[h(y, y') = d_g|h(z, z') = i] \cdot P[h(z, z') = i]$$

$$= \sum_{i=0}^{N-1} p^{(0,i,1)}_{coll} \cdot p^{(i,d_g,1)}_{coll}$$

From Equations 9, 10, and 11,
Thus,

$$p_{col}^{\{0,d_{y},1\}}(f^{(2)}) \leq p_{col}^{\{0,d_{y},1\}}$$ (12)

The other way to look at this is using the bias propagation property (Property 3.3.9). As more and more levels of gates are added the bias will be decreasing and the collision probability approaches $\frac{1}{2}$ for every output bit.

4 **REBEL function family** $R^{2N}_n$

**Definition 4.1 (R^{2N}_n function family).** $R^{N}_n$ is a $I_{2N} \mapsto I_{2N}$ LR-network [8, 4] of $f^{(2)}$ with 4 levels, where $f \in F^{N}_n$.

**4.1 Construction of $R^{2N}_n$**

Let $\kappa \in K$ be the key that chooses the function $f_{\kappa} \in F^{N}_n$. Let $g_{\kappa}$ be the function defined as follows,

$$g_{\kappa}(x|_L \bullet x|_R) = x|_R \bullet (x|_L \oplus f^{(2)}_{\kappa}(x|_R))$$

Let $s$ be a swapping function, such that, $s(x|_L \bullet x|_R) = x|_R \bullet x|_L$.

Then the encryption function $E_{\kappa} \in R^{2N}_n$ is constructed as, $E_{\kappa} = g^{(4)}_{\kappa} \circ s$. The decryption function $D_{\kappa} \in R^{2N}_n$ is same as the $E_{\kappa}$. Figure 5 shows the diagrammatic representation of $E_{\kappa}$ and $D_{\kappa}$.
4.2 Adversary Models

The aim of the adversary in any cryptosystem is to obtain an algorithm to extract the secret (or key) of the family. We define two kinds of adversary models distinguished based on its capabilities.

**Definition 4.2 (function oracle \(O_f\)).** Oracle \(O_f\) performs encryption \((E_\kappa)\) and decryption \((D_\kappa)\) for a randomly chosen key \(\kappa \in \mathcal{K}\). Every query to \(O_f\) is a tuple \((x, e)\) where \(x \in I_{2N}\) and \(e \in \{1, 0\}\). The oracle returns \(y = e \cdot E_\kappa(x) + \bar{e} \cdot D_\kappa(x)\) where \(y \in I_{2N}\).

**Definition 4.3 (unbounded statistics oracle \(O_s\)).** Oracle \(O_s\) has access to the input-output table for the entire key-space \(\mathcal{K}\). Specifically, the model maintained by \(O_s\), \(M(O_s)\), consists of a set of triples \((\{x, y, \kappa\}|x \xrightarrow{\kappa} y)\) where \(x \xrightarrow{\kappa} y\) implies that \(x\) maps to \(y\) with key \(\kappa\). The query to \(O_s\) is an arbitrary set \(Q\) of input-output pairs, potentially chosen adaptively, i.e., \(Q = \{(x, y)|x, y \in I_{2N}\}\) such that \(|Q| = l_q\). The oracle returns a set of keys \(\kappa_Q\) which are consistent with the input-output relations in the query set \(Q\). Let \(|\kappa_Q| = l_r\).

**Definition 4.4 (poly time statistics oracle \(O_s^P\)).** Oracle \(O_s^P\) is similar to the unbounded statistics oracle \(O_s\) except that it only has polynomial time to build its static model of the input-output table with the key-space \(\mathcal{K}\) correlation. For instance, it can choose to develop the correlation for a given key \(\kappa_{i_0}\) with respect to polynomially many input, output pairs. Or it can choose to develop this correlation for a constant number of input-output pairs for polynomially many key values. Hence the cardinality of its model, \(|M(O_s^P)|\), is polynomially bounded. The query to \(O_s^P\) is a set of input-output pairs, \(Q = \{(x, y)|x, y \in I_{2N}\}\) such that \(|Q| = l_q\). The number of queried input-output pairs \(l_q\) is of course polynomially bounded due to the poly time constraint. The oracle returns a set of keys \(\kappa_Q\) from within its model which are consistent with the input-output relations in the query set \(Q\). If none match then an empty set is returned. Let \(|\kappa_Q| = l_r\).

**Definition 4.5 (poly-time static adversary).** A polynomial time static adversary (PTSA) has access to both oracles \(O_f\) and \(O_s\). The adversary performs \(p(N)\) queries to \(O_f\) for a polynomial \(p(N)\). These queries help the adversary build a query set \(Q\) for \(O_s\). The adversary has to decipher the secret (or key) based on the result of \(O_s\).

**Definition 4.6 (poly-time runtime adversary).** A polynomial time runtime adversary (PTRA) has access only to \(O_f\). The adversary performs polynomially in \(N\) many queries to \(O_f\) and based on the input-output relation has to infer the secret (or the key).

4.3 Complexity Analysis (PTRA)

Let \(x, y \in I_{2N}\) and \(y = E_\kappa(x)\). From the construction of \(E_\kappa\),

\[
\begin{align*}
y|_R &= x|_L \oplus f_\kappa^{(2)}(x|_R) \oplus f_\kappa^{(2)}(x|_R \oplus f_\kappa^{(2)}(x|_L)) \\
y|_L &= x|_R \oplus f_\kappa^{(2)}(x|_L) \oplus f_\kappa^{(2)}(y|_R)
\end{align*}
\]

In order to derive any information about the key, which is essentially configurations to the reconfigurable gates, the adversary has to be able to control the input and observe its effect on the output for these component gates. The construction of \(E_\kappa\) ensures that if the input to a gate is controllable then the output is not observable. Hence, the adversary with polynomially many trials can not learn any information about the input output relation of component gates.

The only observable effect of \(f_\kappa\) is its collision. If the adversary chooses a pair \(x|_L, x'|_L\) such that \(f_\kappa^{(2)}(x|_R \oplus f_\kappa^{(2)}(x|_L) \oplus f_\kappa^{(2)}(x'|_R)) = f_\kappa^{(2)}(x|_R \oplus f_\kappa^{(2)}(x'|_L))\) then \(y|_R \oplus x|_L = y'|_R \oplus x'|_L\).

From the Property 3.3.4 of \(F_n^N\) the maximum collision probability occurs when the minimum number of
gates on the top level are exercised. Let, \( x, x' \) be the input to the function \( f_\kappa \), then the collision probability is maximum if the difference (or change) occurs in only one of the group (\( \Delta_\theta(x, x') = 1 \)).

The adversary can determine that two rows of all the \( N \) top level gates collide if such an event can be identified. But the existence of \( f^{(2)}_\kappa(x_{|R}) \) masks input to the second function \( f^{(2)}_\kappa(x'_{|L} \oplus f^{(2)}_\kappa(x_{|R})) \). Let \( E \) be the event such that the adversary can determine that all the \( N \) gates collide in two rows \( r, r' \). This can be considered as partial key extraction, as this information could be used to further reduce the search space in a brute force attack. Let \( x_{|L} \) and \( x'_{|L} \) differ in group \( i \) and let \( g_{\rho_I} \) operator return the \( i^{th} \) group. Then, \( P[E] = P(\text{cause collision in } N \text{ gates}) \cap (r = g_{\rho_I}(x_{|L} \oplus f_\kappa(x_{|R}))) \). Probability of the event \( r = g_{\rho_I}(x_{|L} \oplus f^{(2)}_\kappa(x_{|R})) \) is nothing but \( \frac{1}{2^n} \).

In order to find \( E \) the following experiment is performed. Choose \( x = x_{|L} \oplus x_{|R} \) uniformly at random from the set \( 2N \). Choose \( i \in [1, \frac{N}{n}] \) at random. Then generate \( \left(\frac{2^n}{2}\right) \) pairs of queries to \( O_f \) such that \( x'_{|L} \) differ with \( x_{|L} \) only in bin \( i \). If a collision occurs at the output, the collision could be caused by the first level gates or the subsequent gates. To verify whether top level has caused the collision generate \( N \) pairs at random such that bin \( i \) has the same values as \( x_{|L} \) and \( x'_{|L} \). If the collision is caused by top level gates then every pair will collide. If any one of these pairs does not cause collision then the top level gates have not collided.

Then,
\[
P[E] = \left(\frac{2^n}{2}\right) \cdot \left(\frac{2^{n-1} - 1}{2^{n} - 1}\right)^N \cdot \frac{1}{2^n} < \frac{1}{2^{N-n+1}}
\]

Note that this probability is not affected by the number of trials the adversary is performing, as it is determined by the properties of \( O^{\rho_i}_n \).

Once the adversary has identified a collision between rows \( r \) and \( r' \), the adversary has to guess the actual values of truth table configuration. And the probability of success is \( \frac{1}{2^n} \). Hence, the probability of successfully guessing two rows of truth tables in all the \( N \) gates is bounded by \( \frac{1}{2^{2N-n+1}} \).

\[
P[\text{identifying two rows } r \text{ and } r' \text{ of } N \text{ gates}] < \frac{1}{2^{2N-n+1}}
\]

### 4.4 Statistical Adversary Advantage

In this section, we develop an attack framework for an adversary deploying the statistical oracles \( O_s \) and \( O^{P}_s \).

A few observations are in order. For a given input-output pair \( (x, y) \), the probability that the instantiated key \( \kappa_{\text{inst}} \) collides with another key \( \kappa \), i.e., \( P[E_{\kappa_{\text{inst}}} (x) = E_\kappa (x)] = \frac{1}{2^n} \). This follows from the Property 3.3.2. Thus the expected number of keys that will collide for any given input-output pairs is \( \frac{|G^b_n|^N}{2^{2Nk}} \) and for \( k \) pairs, \( \frac{|G^b_n|^N}{2^{2Nk}} \). Thus with polynomially many input-output pairs (for \( n = 4, k \approx 6 \)) the adversary can get the correct key if the adversary has access to the unbounded statistics oracle \( O_s \). However, the unbounded model \( M(O_s) \) will take unrealistic space and time resources. Hence, we will use the poly bounded oracle \( O^{P}_s \) instead.

Once the oracle is limited to build a model consisting of polynomially many tuples \( (x, y, \kappa) \), it is very unlikely that it can provide the key association for a randomly chosen \( (x, y) \) by the adversary. Note that the adversary can evaluate an arbitrary instance of \( E_{\kappa_{\text{inst}}} (x) \) or \( D_{\kappa_{\text{inst}}} (y) \) through \( O_f \). Hence we do not make any distinction between chosen plaintext or chosen ciphertext attacks. The adversary has to rely upon \( O^{P}_s \) to extract any key association with the input-output pairs. The \( (x, y, \kappa) \) space is very large, \( 2^{2N} \cdot 2^{2N} \cdot |G^b_n|^N \).
Hence, a polynomially bounded oracle has negligible probability that any triples held by it that match on \((x, y)\) pair will also match with respect to the key \(\kappa_{\text{inst}}\). This leads to \(O_s^{P}\)'s inability to help the adversary. The most interesting issue here, then, is what other information can \(O_s^{P}\) provide the adversary, even though it does not contain a consistent \((x, y, \kappa)\) triple. Any such information is useful if it infers a significant fraction of the key bits.

The neighborhood property of the \(E_\kappa\) can be exploited by the model in this respect. The reason such neighborhoods help the adversary is that the oracle does not need to have an exact match for \((x, y)\) in its model any more (which is an unlikely event as we argued). Even if it contains a triple \((x + \Delta x, y + \Delta y, \kappa)\) whose input-output component is from the neighborhood of the query \((x, y)\), it can report back the key of this triple, \(\kappa\), to the adversary. The adversary gains significant information about \(\kappa_{\text{inst}}\) from \(\kappa\) since it is likely that \(\kappa_{\text{inst}} = \kappa + \Delta \kappa\). Depending on the size of the neighborhood, \(\Delta \kappa\), the adversary has a much more tractable problem at hand. The efficiency of this exploration depends on the structure of these neighborhoods. Figure 6 illustrates this issue.

Smaller the neighborhoods, more unlikely it is for the oracle to hold a matching triple in its model from such a neighborhood of the query. Larger neighborhoods will make it more likely that a matching triple from such a neighborhood will be found by the oracle. The adversary will, however, then have to explore the entire neighborhood bubble to determine the full key \(\kappa_{\text{inst}}\). What we will show in this section is that such neighborhoods do not exist in the proposed function family with large probabilities. This makes any experimentation with \(f_\kappa\) equivalent to a random experiment. In other words, success probabilities for an adversary even with a poly bounded statistical oracle are almost as good as they would have been with a random search for the key.

\[\text{From Properties 3.3.8 and 3.3.11 we see that the neighborhood probability gets dispersed with more levels of function compositions. In } E_\kappa \text{ at least 4 levels of function } (f_\kappa) \text{ compositions are there in both the paths (left and right). Hence the neighborhood probabilities estimated for } f_\kappa \text{ will form the upper bound for}\]

![Figure 6: \((x, y, \kappa)\) Neighborhood Bubbles](image-url)
Definition 4.7 (modified poly bounded oracle). We modify the poly bounded $O_s^P$ oracle as follows. On a query $(x, y)$, it returns a triple within its model $(x(i), y(i), \kappa(i)) \in M(O_s^P)$ such that $h(x, x(i)) + h(y, y(i))$ is minimized over $\forall i$.

Lemma 4.1 (statistical adversary advantage). A poly bounded adversary with a poly bounded oracle has probability at most

$$\frac{N^c}{N - N^2^{\frac{N}{2^{2N}} \cdot |G_b|^N}}.$$ 

The proof follows from Equation 10 and from the fact that $(x, y, \kappa)$ space has size $2^{2N} \cdot 2^{2N} \cdot |G_b|^N$.

The main point to note here is that the adversary is almost as well off as it would be in a completely random experiment. Hence, such a polynomially resource bounded oracle adds marginally to an adversary’s capability for the REBEL functions due to lack of any non-uniformity.

4.5 Resilience to Cryptanalysis

In this section we investigate the resilience of REBEL functions towards the well known cryptanalysis methods. First we will show the class of attacks which use the invariance property of system, i.e., the idea of these attacks is to find the properties of the system which are not dependent or least dependent either on the secret or the input.

4.5.1 Linear Cryptanalysis

Linear cryptanalysis is a general form of cryptanalysis based on finding affine approximations to the cipher function. The technique [10] has been applied to attack ciphers like FEAL [11] and DES [12]. Linear cryptanalysis exploits the high probability of occurrences of linear expressions involving plaintext, ciphertext, and sub-key bits. This attack becomes possible on the conventional cipher function design as the key bits are primarily XOR’ed with round inputs. And linear approximations of known components (S-boxes) in the system further aid the analysis. In the case of REBEL none of these required conditions are available. Every gate (component gates of tree) is chosen by key and hence no linear approximation is possible.

4.5.2 Differential Cryptanalysis

Differential cryptanalysis [2] exploits the property of difference being propagated from input to the output of the cipher function. This attack again requires the properties of the known components in the system (S-boxes) in order to estimate the difference propagation probabilities. In REBEL such an analysis is not possible as there are no known components in the system. A variant to this attack is impossible difference attack [1] which again uses the principle of identifying differences that does not propagate from input to output.

4.5.3 Boomerang Attack

This attack [15] relies on the difference analysis of round function properties and existence of some block in the system which is independent of the input to cipher function. This can be thought of as meet-in-the-middle version of differential cryptanalysis. Again REBEL is resistant as there are no blocks (gates) in the system that is either independent of key or the input.
4.5.4 Sliding Attack

This attack [3] exploits the weakness of the round functions. It assumes that given two pairs \( P, C \) and \( P', C' \) such that \( P' = f(P) \) and \( C' = f(C) \) then the round function can be deciphered or at least significant bits of the keys can be extracted. These attacks once again uses the property of round functions being built using some known components (S-boxes) and key bits being used only in XOR operations.

In REBEL the round function constitute \( f^{(2)}_k \). As shown in Section 3.3.11 the round function can not be deciphered given a polynomial sized set of input output pairs. Hence sliding attack is also ineffective against REBEL.

4.6 Pseudorandomness

A permutation generator informally is pseudorandom if no polynomial time probabilistic adversary can distinguish it from a random permutation. Pseudorandomness is a desirable characteristic since it rules out predictable models for an adversary. In this section, we show that REBEL is a pseudorandom permutation generator.

We will use the Luby, Rackoff [8] result to establish the pseudorandom permutation property. Luby and Rackoff show that if the constituent function in a LR network is pseudorandom then the resulting permutation is also pseudorandom. Since REBEL is constructed as an LR network with the function \( f^{(2)} \) where \( f \in F_n^N \), it will suffice to show that \( f \in F_n^N \) is a pseudorandom function. In the following, we will establish that it is difficult to distinguish between a function \( f \in F_n^N \) and a random function only with polynomially many observations. Our notion of pseudorandomness closely resembles that of Goldreich, Goldwasser, Micalli [6] in as much as we show that the pseudorandom function is indistinguishable from a random one under all efficient black-box attacks. The adversary can observe the output of the function at polynomially many input instances. Specifically, we demonstrate the indistinguishability with respect to the key property of balance.

The key issue for an adversary is to be able to distinguish a random function from the universe of the \( 2^{2N} \) Boolean \( N \)-bit functions \( U^N \) from a function from \( G_N^N \). Note that technically we are dealing with functions of \( n \) output bits, not Boolean, and with \( F_n^N \) and not \( G_N^N \). However, as we will see, both are reasonable simplifications. The following experiment describes the pseudorandomness requirement. The following steps are repeated polynomially many \( \rho(N) \) times. (1) Oracle chooses a function \( f \) either from \( F_n^N \) or from \( U^N \) with equal probability; (2) Adversary specifies an input vector \( \vec{x}_i \); (3) The oracle returns \( \vec{y}_i = f(\vec{x}_i) \); (4) The adversary guesses whether \( \vec{y}_i \) is output by a function from \( F_n^N \) or from \( U^N \). Let \( p_s \) denote the probability with which the adversary makes the correct determination. The adversary’s advantage is captured by \( |p_s - 1/2| \).

We need to show that the adversary’s distinguishing probability \( |p_F(f) - p_U(f)\) is asymptotically 0. In other words, a function \( f \in F_n^N \) appears like a function in \( U^N \) to the adversary with asymptotic probability 1. To that end, we need to characterize a random function from the universe \( U^N \), specifically with respect to the observability of its balance. In the following, we claim that the average function in \( U^N \) is balanced within \( 2^{N/2} \) (\( 2^{N/2} \)-balanced), i.e., \( ||\{\vec{x}_i.s.t.f(\vec{x}_i) = 1\}|| - ||\{\vec{x}_i.s.t.f(\vec{x}_i) = 0\}|| \leq 2^{N/2} \) for an average \( f \in U^N \). Note the implicit definition in the preceding for a \( k \)-balanced function \( f \) which means that \( ||\{\vec{x}_i.s.t.f(\vec{x}_i) = 1\}|| - ||\{\vec{x}_i.s.t.f(\vec{x}_i) = 0\}|| \leq k \).

Lemma 4.2 (universe balance): Let \( U^N_k \) denote the set of functions in \( U^N \) that are \( k \)-balanced. For a constant \( .5 < c_0 < 1 \), there exists a constant \( c_1 \) such that the fraction of \( c_1.2^{N/2} \) functions is at least \( c_0 \), i.e., \( |U^N_{c_1.2^{N/2}}|/|U^N| \geq c_0 \). This implies that majority of functions in \( U^N \) have balance lower than the order of \( 2^{N/2} \).
Proof. The functions in \( U^N \) can be classified into \( 2^N \) disjoint sets \( U^N(i) \) of cardinality \( \binom{2^N}{i} \) with exactly \( i \) 0's in the truth table. The set \( U^N(2^N-1) \) with \( \binom{2^N}{2^N-1} \) functions is a balanced set. The sets \( U^N(2^N-1-k) \) and \( U^N(2^N-1+k) \) are \( k \)-balanced. The key point to note here is that the function \( \binom{2^N}{i} \) is maximized at the midpoint \( i = 2^N-1 \). It drops off slowly around the midpoint for \( i \approx 2^N-1 \pm 2^{N/2} \). In fact, almost all of the \( 2^{2N} \) functions of \( U^N \) belong to the \( \pm 2^{N/2} \) neighborhood of the midpoint. This will establish that the average imbalance in \( U^N \) is no worse than the order of \( 2^{N/2} \). We will first estimate the cardinality of each of these sets.

We start with the midpoint \( U^N(2^N-1) \), the subset of balanced functions. Its cardinality is \( \binom{2^N}{2^N-1} \), and hence it accounts for \( \binom{2^N}{2^N-1}/2^{2N} \) fraction of all functions in \( U^N \). \( \binom{2^N}{2^N-1} \) equals \( 2^{N(1/2 - 1)} \), which can be rewritten as follows with Stirling’s approximation (Gosper’s version) [5].

\[
\binom{2^N}{2^N-1} = \frac{(2^N/e)^{2^N-1} \sqrt{(2^N+1/3)\pi} (2^N-1)}{\sqrt{2^N+1/3)} \pi}.
\]

This can be simplified into \( 2^{2N}/(\sqrt{\pi} \cdot 2^{N/2}) \), which implies that fraction of functions from \( U^N \) contained in \( U^N(2^N-1) \) is \( 1/(\sqrt{\pi} \cdot 2^{N/2}) \). Hence, if all the sets \( U^N(2^N-1 \pm i) \) were to contribute a similar number of functions, we need to accumulate order of \( 2^{N/2} \) sets, up to \( U^N(2^N-1 \pm 2^{N/2}) \) to collect more than half of the functions in \( U^N \). We show that in the following.

Let us estimate the cardinality of \( U^N(2^N-1-l) \) for \( 1 \leq l \leq 2^{N/2} \), which is given by \( \binom{2^N}{2^N-1-l} \). This is approximated by

\[
\binom{(2^N-1-l)}{2^N} \cdot \frac{(2^N/e)^{2^N-l} \sqrt{(2^N-1-l)+1/3)\pi} ((2^N-1-l)/e}^{2^N-l} \pi \sqrt{2^N-1-l} \sqrt{2^N+l+1/3)\pi}.
\]

This simplifies to \( 2^{2N}/(e^{2N}) \) which further equals

\[
(2^N)_{2^N}^{(2^N-1-l)}(1-\l/2^N) |^{(2^N-1-l)+1/3)\pi} ((2^N-1-l)/e}^{2^N-l} \pi \sqrt{2^N-1-l} \sqrt{2^N+l+1/3)\pi}.
\]

Using the Taylor’s series approximation for \( e^x \approx (1 + x) \) for \( x << 1 \) since \( l/2^N << 1 \) gives

\[
\binom{(2^N-1-l)}{2^N} \approx \binom{2^N}{2^N} \cdot e^{2^N(l-1)/2^{N+1}+1/3)\pi} \sqrt{2^N/2^{N+1}+1/3)\pi}.
\]

If we choose to cumulate the functions over \( 2^{N/2} \) of the sets \( U^N(2^N-1-l) \) for \( 1 \leq l \leq 2^{N/2} \), the expected fraction of covered functions approaches 1 in the limit. This fraction has the form \( \int_1^{2^{N/2}} \frac{1}{e^{2^N(l-1)/2^{N+1}+1/3)\pi} \sqrt{2^N/2^{N+1}+1/3)\pi} \), which is best performed as a Taylor series expansion. The equivalent expression is

\[
\sum_{l=1}^{2^{N/2}} \left( 1 - \frac{l^2}{2^{2N} \pi} \right)
\]

which approximates to \( 5/2^{2N}/6 \). Hence the fraction of functions covered by the sets \( U^N(2^N-1-l) \) for \( 0 \leq l \leq 2^{N/2} \) is approximately \( 1/\sqrt{2^N/2^{N+1}+1/3)\pi} \approx .9403 \). Note that for any other value of \( 0.5 < c_0 < 1 \), the number of sets \( U^N(2^N-1 \pm l) \) included can be tweaked such that \( 0 < 0.5 < 2^{N/2} \) to ensure that \( 1 + 2 * 2^{N/2}/6 \approx .9403 \). In such a case, the imbalance of these functions is at most \( c_1 \cdot 2^{N/2} \), where \( c_1 \) is a constant.

The next step is to assess that given that the imbalance in a random function is at most of the order of \( 2^{N/2} \), how likely it is that an adversary will be able to detect this imbalance with polynomially many trials. We show in the following that a function \( f \in F_1^N \) appears like a function in \( U^N \) with polynomially many observations with asymptotic probability 1, and vice versa.

Lemma 4.3 (function distinguishability). Let a function \( f \in F_1^N \) be evaluated \( p(N) \) times independently on \( p(N) \) unique input vectors \( \bar{x}_i \) for \( p(N) \) a polynomial in \( N \). The resulting experiment is \( S_p(N) = X_1 + \cdots + X_p(N) \) where each \( X_i \) contributes a \( 1 \) (a 0) with probability \( p = 1/2 \) \( (q = 1-p = 1/2) \). The function \( f \) will appear to be a function from \( U^N \) if \( S_p(N) \geq p(N) \cdot \left[ \frac{1}{2} + \frac{1}{2^{N/2}} \right] \). This probability
\[ Pr\left[S_{p(N)} \geq p(N) \cdot \left( \frac{1}{2} + \frac{1}{2^{N/2}} \right) \right] \approx (1 - p(N)/2^{N-2}). \]

**Proof.** The proof is based on Chernoff bound [5] which states that \[ Pr\left[S_{p(N)} \geq p(N) \cdot \left( \frac{1}{2} + \frac{1}{2^{N/2}} \right) \right] \approx e^{-p(N)\left[a \log(a/p) + b \log(b/q)\right]} \text{ where } a = 1/2 + 1/2^{N/2} \text{ and } b = (1 - a) = 1/2 - 1/2^{N/2}. \] This probability is given by \( e^{-p(N)K(a, p)} \). Let us estimate \( K(a, p) = [1/2 + 1/2^{N/2}] \cdot \log(2 * [1/2 + 1/2^{N/2}]) + [1/2 - 1/2^{N/2}] \cdot \log(2 * [1/2 - 1/2^{N/2}]) \). This equals \([1/2 + 1/2^{N/2}] \cdot \log(1 + 1/2^{(N/2) - 1}) + [1/2 - 1/2^{N/2}] \cdot \log(1 - 1/2^{(N/2) - 1})\). We will approximate \( \log(1 + x) \) by \( x \) based on the Taylor’s series expansion of \( \log(1 + x) \) for \( x << 1 \). We then get \([1/2 + 1/2^{N/2}] \cdot 1/2^{(N/2) - 1} + [1/2 - 1/2^{N/2}] \cdot (-1/2^{(N/2) - 1})\). This implies that \[ Pr\left[S_{p(N)} \geq p(N) \cdot \left( \frac{1}{2} + \frac{1}{2^{N/2}} \right) \right] \approx e^{-p(N)/2^{N-2}} \approx (1 - p(N)/2^{N-2}). \] The proof to show that a function in \( U^N \) looks similar to a function in \( F_n^N \) will be similar. \qed

Lemma 4.3 establishes that the adversary’s advantage is of the order of \( p(N)/2^{N-2} \) which implies that REBEL functions are pseudorandom. The LR construction based on the functions from \( F_n^N \) to derive \( R_n^{2N} \) then becomes pseudorandom permutation by Luby, Rackoff proof [8].

**Theorem 4.1.** A function \( f \in R_n^{2N} \) is a pseudorandom permutation.

### 4.7 Implementation:

The implementation of REBEL requires the usage of \( n \)-to-1 reconfigurable gates. These reconfigurable gates are implemented as \( 2^n \)-to-1 multiplexers and hence the size (as well as delay) increases exponentially with the increase in \( n \). The practicable value of \( n \) is 4, as lesser than that will not have sufficiently rich gate space, and more than that will have very high area and delay penalty. Table 1 lists the properties two instances of the REBEL function family.

<table>
<thead>
<tr>
<th>Property</th>
<th>( R_4^{32} )</th>
<th>( R_4^{128} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>(</td>
<td>G^N_4</td>
<td>)</td>
</tr>
<tr>
<td>key size (bits)</td>
<td>256</td>
<td>1024</td>
</tr>
<tr>
<td>(</td>
<td>K</td>
<td>)</td>
</tr>
<tr>
<td>( \delta_c )</td>
<td>( \left( 1 - \left( \frac{8}{15} \right)^2 \right)^{16} )</td>
<td>( \left( 1 - \left( \frac{8}{15} \right)^3 \right)^{64} )</td>
</tr>
<tr>
<td>( P_{coll} )</td>
<td>( \approx \frac{1.9}{2^{16}} )</td>
<td>( \approx \frac{1.076}{2^{64}} )</td>
</tr>
</tbody>
</table>

### 5 Conclusion

In this paper, we have provided a novel design approach towards constructing block cipher functions using reconfigurable gates. A fixed logic network of reconfigurable gates serves as the block cipher schema. The specific configurations loaded into these gates constitute the secret. Each bit of the ciphertext is derived independently from a logic tree. This endows randomness or lack of correlation between these bits offering many desirable cryptographic properties. This lack of coordination, however, also has potential for a high
frequency of collisions. We provide detailed analysis of various properties of REBEL function family. For instance, collision probabilities are extremely low. We are also able to show that REBEL functions are pseudorandom permutations. Moreover, we believe that these ciphers can deliver 100+ GB/second encryption throughput based on our experience with a reconfigurable hash function implementation [9].

References


