On Linear Approximation of Modulo Sum

Alexander Maximov

Department of Information Technology
Lund University, Box 118, SE–22100 Lund, Sweden
e-mail: movax@it.lth.se

Abstract. The general case for a linear approximation of the form "$X_1 + \cdots + X_k \mod 2^n$" $\rightarrow$ "$X_1 \oplus \cdots \oplus X_k \oplus N$" is investigated, where the variables and operations are n-bit based, and the noise variable N is introduced due to the approximation. An efficient and practical algorithm of complexity $O(n \cdot 2^{3(k-1)})$ to calculate the probability $\Pr\{N\}$ is given, and in some cases it can be reduced to $O(2^{k-2})$.

1 Introduction

Linear approximations of nonlinear blocks in a cipher is a common tool for cryptanalysis. One of the most typical approximations is the substitution of the arithmetical sum modulo 2^n ($\boxplus$) with the XOR-operation ($\oplus$) of the input variables. We introduce a noise variable N and write: $X_1 \boxplus \cdots \boxplus X_k = X_1 \oplus \cdots \oplus X_k \oplus N$. For a distinguishing attack the bias of a linear combination of noise variables can be calculated if their distributions are known. For the considered approximation the distribution of N can be calculated in two ways:

I. for $X_1 = 0 \ldots 2^n - 1$ $\quad\leftarrow O(2^{k \cdot n})$
$\quad\ddots$
$\quad$ for $X_k = 0 \ldots 2^n - 1$
$\quad\quad$ $\text{Dist}_N[(X_1 \boxplus \cdots \boxplus X_k) \oplus$
$\quad\quad\quad$ $(X_1 \oplus \cdots \oplus X_k)]$++;

or

II. for $C = 0 \ldots 2^n - 1$ $\quad\leftarrow O(c \cdot 2^n)$
$\quad$ Dist_N[C]=ProbOfN(C);

where the function ProbOfN(C) calculates the corresponding probability (see Section 2). Note that we deal with integer-valued distribution tables, i.e., $\Pr\{N = C\} = \text{Dist}_N[C]/2^{k \cdot n}$.

2 The Function ProbOfN(C)

Let $C = \overline{c_n \ldots c_2 0}$ (note that $\Pr\{N = \overline{c_n \ldots c_2 1}\} = 0$). Then:

$$\text{ProbOfN}(C) = (1\ 1 \ldots 1) \times \prod_{i=n}^{2} \mathbf{T}_{c_i} \times \mathbf{S_0},$$

where $\mathbf{T_0}$, $\mathbf{T_1}$, and $\mathbf{S_0}$ are fixed matrices. The algorithm to construct the matrices $\mathbf{T_0}$, $\mathbf{T_1}$, and $\mathbf{S_0}$ is given below.

Initialization:
$\quad$ $\mathbf{S_0} = (\mathbf{0})$ - is of size $(2^{k-1} \times 1)$
$\quad$ $\mathbf{T_0} = \mathbf{T_1} = (\mathbf{0})$ - is of size $(2^{k-1} \times 2^{k-1})$

Algorithm 1: $\mathbf{S_0}$ – *construction*
$\quad$ 1. for $X = 0$ to $2^k - 1$
$\quad$ 2. $\quad$ $\mathbf{S_0}[\lfloor \frac{\#X}{2} \rfloor] += 1$

Algorithm 2: $\mathbf{T_0}$, $\mathbf{T_1}$ – *construction*
$\quad$ 1. for $C = 0$ to $2^{k-2} - 1$
$\quad$ 2. $\quad$ for $X = 0$ to $2^k - 1$
$\quad$ 3. $\quad\quad$ $\mathbf{T_0}[C + \lfloor \frac{\#X}{2} \rfloor][2C] ++$,
$\quad$ 4. $\quad\quad$ $\mathbf{T_1}[C + \lfloor \frac{\#X+1}{2} \rfloor][2C+1] ++$;

where $\#X$ is the *Hamming weight* of X.

3 Example

Assume that $n = 5$ and $k = 3$, i.e., $N = (X_1 \boxplus X_2 \boxplus X_3) \oplus (X_1 \oplus X_2 \oplus X_3)$. Then:

$$\mathbf{T_0} = \begin{pmatrix} 4&0&0&0 \\ 4&0&4&0 \\ 0&0&4&0 \\ 0&0&0&0 \end{pmatrix} \mathbf{T_1} = \begin{pmatrix} 0&1&0&0 \\ 0&6&0&1 \\ 0&1&0&6 \\ 0&0&0&1 \end{pmatrix} \mathbf{S_0} = \begin{pmatrix} 4 \\ 4 \\ 0 \\ 0 \end{pmatrix}.$$

Let $C = \overline{10110}$, then ProbOfN(C)= $(1\ 1\ 1\ 1) \times \mathbf{T_1} \times \mathbf{T_0} \times \mathbf{T_1} \times \mathbf{T_1} \times \mathbf{S_0}$, and $\Rightarrow \Pr\{N = \overline{10110}\} = 1536/2^{3 \cdot 5} = 0.046875$.

4 Optimization Ideas

If n is not very large, say $n = 32$ bits, then optimization can be done in the following way. Represent $C = \overline{AB0}$, where $A = \overline{c_{32} \ldots c_{16}}$ and $B = \overline{c_{15} \ldots c_2}$. Then create two tables of vectors: $R_{Left}[A] = (1\ \ 1 \ldots 1) \times \prod_{i=32}^{16} \mathbf{T}_{c_i}$ and $R_{Right}[B] = \prod_{i=15}^{2} \mathbf{T}_{c_i} \times \mathbf{S_0}$, for all A and B. Then the probability $\Pr\{N = C\}$ is just a scalar product $R_{Left}[\overline{c_{32} \cdots c_{16}}] \times R_{Right}[\overline{c_{15} \cdots c_2}]$, and the time complexity is $O(2^{k-2})$. This idea of partitioning can be extended to larger n as well.