Analysis of ARX Functions: Pseudo-linear Methods for Approximation, Differentials, and Evaluating Diffusion

Kerry A. McKay[*] and Poorvi L. Vora[**]

The George Washington University, Washington DC 20052, USA
kerry@gwmail.gwu.edu and poorvi@gwu.edu

Abstract. This paper explores the approximation of addition mod 2^n by addition mod 2^w, where $1 \leq w \leq n$, in ARX functions that use large words (e.g., 32-bit words or 64-bit words). Three main areas are explored. First, *pseudo-linear approximations* aim to approximate the bits of a w-bit window of the state after some rounds. Second, the methods used in these approximations are also used to construct truncated differentials. Third, branch number metrics for diffusion are examined for ARX functions with large words, and variants of the differential and linear branch number characteristics based on pseudo-linear methods are introduced. These variants are called *effective differential branch number* and *effective linear branch number*, respectively. Applications of these approximation, differential, and diffusion evaluation techniques are demonstrated on Threefish-256 and Threefish-512.

1 Introduction

Modern block ciphers are expected to be resilient to linear cryptanalysis [18], which relies on the existence of linear distinguishers: probabilistic linear relationships over $\mathbb{Z}_2$, among plaintext, ciphertext, and key bits. In order to protect against effective linear cryptanalysis, some block ciphers have employed a combination of addition modulo 2^n, rotation and exclusive-or (ARX), preventing the existence of linear relations *over* $\mathbb{Z}_2$ among *single* bits. In this paper, we present *pseudo-linear cryptanalysis*, which approximates *strings of bits* using addition in underlying structures *other than* $\mathbb{Z}_2$ and is hence a more effective analytical tool for ARX designs. We illustrate the use of pseudo-linear analysis by applying it to the Threefish block cipher of the Skein design [9], one of five finalists in the SHA-3 competition. The paper also presents metrics for the evaluation of diffusion in ARX designs, to predict vulnerability to pseudo-linear cryptanalysis. In keeping with the tradition of linear cryptanalysis and the terminology common in the cryptanalytic community, in this paper, the words "linear" and "linearity" will refer to linearity over $\mathbb{Z}_2$.

ARX designs are simple, efficient and easy to implement. Desktop computers easily support 32-bit words, and many newer architectures support 64-bit words, all of which leads to cheap and efficient processing. In addition, the use of addition modulo 2^n reduces the memory footprint that may otherwise be used for substitution box table lookups. Computing the non-linear function in memory is also a means of defense against memory inspection techniques that look for certain structures in memory, such as lookup tables, to identify the use of cryptography. ARX-based block ciphers can also be used to design secure hash functions; in fact, two of the five finalists in the SHA-3 competition were ARX-based. Distinguishers of the block cipher could be used in hash function analysis to discover preimages and collisions. Given their flexibility and efficiency, it is expected that ARX designs will continue to be popular. However, the ARX approach has not been analyzed extensively and any general framework for analyzing ARX functions will have impact on future cipher designs.

This paper presents an approach to analyzing and measuring the security of ARX-based block ciphers. Its focus is on the analytical approach in contrast to the "breaking" of a single cipher. The key idea in the approximation is to examine a window (grouping of contiguous bits) of size w for $w < n$. We make the following simple observations. First, addition modulo 2^n on the window can be approximated by addition modulo

[*] Work supported in part by the National Science Foundation Scholarship for Service Program, grant DUE-0621334, and National Science Foundation grant CCF 0830576

[**] Work supported in part by National Science Foundation grant CCF 0830576

2^w. Second, this addition gives a perfect approximation if the carry into the window is estimated correctly. The probability of correctness of the approximations depends exclusively on the probability distribution of the carry, and this probability is independent of w; in fact, for uniformly distributed addends it is $\frac{1}{2} + 2^{-i-1}$, where $i \geq 0$ is the position of the least significant bit in the window. Third, the probability of correctness for a random guess of the value of the window decreases exponentially with w; it is $\frac{1}{2^w}$. Hence, the bias of the approximation (the difference between the probability of correctness of our approximation and that of a random guess) increases with w.

When $w = 1$, pseudo-linear cryptanalysis provides a correctness probability exactly that of linear cryptanalysis with a bias of 2^{-i-1}, which is small for large i (larger values of i imply the bit being approximated is more significant). It is hence correct to assume that ARX ciphers are not very vulnerable to traditional linear cryptanalysis. On the other hand, pseudo-linear cryptanalysis for large w and small i results in a correctness probability greater than $\frac{1}{2}$ for a string of w bits, providing considerable advantage over guessing at random.

This paper presents pseudo-linear cryptanalysis and simple results about the approximations in terms of w. We show how approximations can be combined over a number of rounds. Because the estimate of the carries significantly affects the output, one may compensate for mispredicted carries by adding an offset to the approximated string and looking at an interval around the string. This paper also provides extensions of the *branch number* diffusion metric used in the design strategy of the Advanced Encryption Standard (AES) to better analyze ARX functions that use 32 and 64-bit word sizes. The application of pseudo-linear analysis and extended diffusion metrics is demonstrated on Threefish-512, the ARX block cipher found in the main submission of SHA-3 finalist Skein [9]. It is worth noting that pseudo-linear cryptanalysis presumes one of the weakest adversaries in the literature: the adversary has access to plaintext/ciphertext pairs, but has no power over the selection of plaintext, ciphertext, or keys, and does not assume any relationships between samples.

This paper is organized as follows. Section 2 presents related work. Section 3 presents pseudo-linear cryptanalysis and section 4 demonstrates links with differential cryptanalysis. Section 5 presents diffusion metrics and section 6 applies all results to the Threefish cipher of the Skein hash function entry to the SHA-3 competition. The conclusion is presented in section 7. Some of this work has been presented at the second SHA-3 conference [19] and a pre-reviewed version of that paper is available as a technical report [20]. In addition to the material in that paper, this paper also presents pseudo-linear results on Threefish-512, links between pseudo-linear and differential cryptanalysis with results on Threefish-512, and an exploration of how the branch number metric applies to ARX ciphers that perform operations on large words.

Notation: Throughout this paper, the following notation is used.

$\boxplus$ addition modulo 2^n, where n is the word size in bits
$\boxplus_w$ addition modulo 2^w, where $1 \leq w \leq n$
$\oplus$ exclusive-or
$\lll_r$ r-bit left word rotation
$\ggg_r$ r-bit right word rotation
x_i^j word i of state x at round j
x_i the bit at position $0 \leq i < n$ of word x
$x(i)$ bits $x_i, \ldots, x_{(i+w-1) \bmod n}$ of word x, for $i \leq 0 < n$ and $1 \leq w \leq n$

2 Related Work

Linear cryptanalysis consists of approximating a function using linear expressions. This method was created to attack DES, and as such, was designed for approximating functions using exclusive-or, or addition modulo 2. An approximation of the relationship among plaintext, ciphertext and key bits would be of the form $p_1 \oplus p_2 \oplus \ldots \oplus p_n \oplus c_i \oplus \ldots \oplus c_m = k_1 \oplus \ldots \oplus k_s$, where each p_i is a bit of plaintext, each c_i is a bit of ciphertext, and each k_i is a bit of key. Results can be improved using multiple linear approximations to perform linear cryptanalysis [11]. In recent years, the concept of linear cryptanalysis has been generalized to encompass non-binary ciphers [2].

The use of addition mod 2^n in cryptographic algorithms has inspired several researchers to study the linear properties involved in integer addition. The carry function has been shown to be biased, based on position [29]. In addition to position, addition also has bias properties based on the number of inputs [23]. The probability distribution of the carry function in integer addition with an arbitrary number of inputs has been investigated[24]. The carry distribution of two neighboring bits during addition modulo 2^{32} – has also been studied[5]. Nyberg and Wallén have made considerable contributions in linear approximations of addition and understanding the carry function [23][26][27][28][22][21].

Differential cryptanalysis exploits relations between pairs of inputs, and often requires the adversary to have power over choosing the plaintext or choosing the ciphertext. That is, the adversary chooses two message, M and M', that are related by some known value, $\varDelta$, such that $M' = M \oplus \varDelta$. A generalization of a differential characteristic called a truncated differential characteristic [14] requires a difference to be satisfied in a subset of state bits, rather than the entire state. Higher order differentials [14] do not restrict differences to linear relationships. Higher order differential cryptanalysis extends analysis beyond the first order derivative[15].

More complex differential attacks, such as boomerang attacks, have been described[25]. Related-key boomerang attacks have been applied to Threefish, the block cipher in Skein's [8] compression function [4][1]. Impossible differential attacks use knowledge of differences that can never happen to learn secret information.

Rotational cryptanalysis [13] is a recent differential technique based on two properties that always hold, independent of the cipher.

$$Pr[\overrightarrow{X \oplus Y} = \overrightarrow{X} \oplus \overrightarrow{Y}] = 1$$

$$Pr[\overrightarrow{X \boxplus Y} = \overrightarrow{X} \boxplus \overrightarrow{Y}] = \frac{1}{4}(1 + 2^{r-n} + 2^{-r} + 2^{-n})$$

where $\overrightarrow{X}$ is a right rotation of n-bit word X by r positions and $\boxplus$ is addition modulo 2^n.

This attack requires a very strong adversary who has the ability to manipulate key and data pairs. As such, the meaning and applicability of this type of attack is not clear. If a block cipher is part of a hash function, where keys are derived from some portion of the state, this could be a valid attack. As an attack on a standalone block cipher, however, the model may not be as valid.

A relationship between linear and differential cryptanalysis has been shown [3]. Linear and differential methods have been combined into a single attack, called Differential-Linear Cryptanalysis[16].

Mod n cryptanalysis[12] is a different sort of partitioning attack that exploits properties in ciphers based on addition and rotation. One key observation is that addition modulo 2^n, where n is the word length in bits, can be approximated by a smaller word size. A mod 3 attack was shown for RC5P where the distribution of mod 3 values was distinguishable from a uniform distribution using a χ^2 test.

Branch number[6][7] is a metric that gives a lower bound on the diffusion of a transformation. It is part of the wide-trail strategy that was used to design the Rijndael block cipher – the winner of the Advanced Encryption Standard (AES) competition — and is now a widely adopted standard. The metric has been defined for linear and differential trails over any partition [6], as well as a specific type of partition used in AES[7].

3 Pseudo-Linear Cryptanalysis

Pseudo-linear cryptanalysis aims to overcome limitations of traditional linear cryptanalysis by approximating addition modulo 2^n with addition modulo $2^w, 1 \leq w \leq n$. This differs from linearization (used by linear cryptanalysis), which approximates the addition with exclusive-or i.e. addition modulo 2.

More specifically, pseudo-linear approximations use addition modulo 2^w and exclusive-or over w-bit strings of contiguous bits (*windows*). Let $part(x, s, e)$ represent bits of word x in the range $[s, e)$, where $0 \leq s < n$, $0 \leq e < n$, and the least significant bit is at index 0. Note that if $e < s$, the range will include bits both above s and below e.

Consider the scenario presented in figure 1, where the adversary desires to approximate the middle byte of word $z = \texttt{0x68ACF0}$ in terms of addends $x = \texttt{0x012345}$ and $y = \texttt{0x6789AB}$. The approximation $part(z, s, e) = part(x, s, e) \boxplus_w part(y, s, e)$ is true because the carry into the window is predicted correctly. However, if $x = \texttt{0x0123FF}$, the approximation would not be correct because of the carry into the middle byte.

$$\begin{array}{r} \texttt{0x012345} \\ \boxplus\ \texttt{0x6789AB} \\ \hline \texttt{0x68ACF0} \end{array} \longrightarrow \begin{array}{r} \texttt{0x002300} \\ \boxplus\ \texttt{0x008900} \\ \hline \texttt{0x00AC00} \end{array}$$

Fig. 1. Approximation of a window of $z = x \boxplus y$, where $x = \texttt{0x012345}$ and $y = \texttt{0x6789AB}$.

3.1 Addition Windows: Simple Analytical Results

Properties of addition, such as position-based bias, can be leveraged to increase the accuracy of carry prediction. A simple case to analyze is the addition of n-bit words that are sampled from a uniformly random distribution.

Consider two n-bit words, x and y, selected uniformly at random, and a window size $w < n$.

Lemma 1. *Let $0 \leq s < s+w < n$. Then $\Pr[part(x \boxplus y, s, s+w) = part(x, s, s+w) \boxplus_w part(y, s, s+w)] > \frac{1}{2}$.*

Proof. If there is no carry into bit s, then the equation is true. If there is a carry into bit s, then the equation is false. For the addition of two s-bit integers, there are $2^{2s-1} + 2^{s-1}$ of 2^{2s} operand combinations that will not produce a carry in the output. This is because for each value of $x \in \{0, \ldots, 2^n - 1\}$, there are $2^s - x - 1$ possible values of $y \in \{0, \ldots, 2^n - 1\}$ such that $x + y$ does not produce a carry, giving $\sum_{i=1}^{2^s} 2^s - i$ possible addend combinations that do not produce a carry. Therefore the probability that the carry into bit s is 0 is $\frac{1}{2} + 2^{-s-1}$, which agrees with the bias presented in [29].

Lemma 2. *$\Pr[part(x \boxplus y, 0, w) = part(x, 0, w) \boxplus_w part(y, 0, w)] = 1$.*

Proof. Because this is the least significant window, there are no carry bits into the sum. Therefore the approximation is always correct.

Corollary 1 *Let $0 \leq s < n$ and $(s + w) \bmod n < s$. $\Pr[part(x \boxplus y, s, (s + w) \bmod n) = part(x, s, (s + w) \bmod n) \boxplus_w part(y, s, (s + w) \bmod n)] > \frac{1}{2}$.*

Proof. The main difference in this case is that the window may wrap around from the higher end of the n-bit word to the lower end. If the window does not wrap around, this is equivalent to Lemma 1. If it does, then there is one carry that is not propagated; namely the carry out of the n-bit sum. Split the window in two: $part(x, s, n) \boxplus_w part(y, s, n)$ and $part(x, 0, (s + w) \bmod n) \boxplus_w part(y, 0, (s + w) \bmod n)$. The first sum is true by Lemma 1 and the second is true by Lemma 2.

Corollary 2 *$\Pr[part(x \boxminus y, s, (s+w) \bmod n) = part(x, s, (s+w) \bmod n) \boxminus_w part(y, s, (s+w) \bmod n)] > \frac{1}{2}$.*

Proof. Let $x \boxminus y = z$. Then $y \boxplus z = x$, and $part(y \boxplus z, s, (s + w) \bmod n) = part(y, s, (s + w) \bmod n) \boxplus_w part(z, s, (s+w) \bmod n)$ with probability greater than $\frac{1}{2}$ by Corollary 1. Then we have $part(x, s, (s+w) \bmod n) = part(y, s, (s + w) \bmod n) \boxplus part(z, s, (s + w) \bmod n)$, which is the same as $part(x, s, (s + w) \bmod n) \boxminus part(y, s, (s + w) \bmod n) = part(z, s, (s + w) \bmod n)$, with probability greater than $\frac{1}{2}$.

Lemma 3. *$\Pr[part(x \oplus y, s, (s + w) \bmod n) = part(x, s, (s + w) \bmod n) \oplus part(y, s, (s + w) \bmod n)] = 1$.*

Proof. This follows directly from the bitwise nature of exclusive-or.

Using these equations, one can easily approximate windows derived from a single addition. However, this is not a realistic scenario for approximations over an iterated cipher. After the first approximated addition, the input distribution of all subsequent additions changes. In particular, the distribution is dependent on the input distribution of the operand bits preceding the target window. To see how the distribution may change, consider the equation $z = x \boxplus y$. Suppose we are interested in approximating the window of z starting with the third least significant bit, and assume that x and y are selected at random from the uniform distribution. Since the carry into the addition window depends on the preceding bits, the distribution of all bits preceding the target window are important. These bits are also considered unknown if they are not a part of the approximation.

Suppose there were no carry into the addition window. Ten of the sixteen possible input combinations produce this result. The possible values for (z_0, z_1) are found in table 1, along with the probability of occurrence and the operands that produce each of the values. The distribution is clearly no longer uniform, and is biased towards larger values.

$z_1\ z_0$	$x_1\ x_0$	$y_1\ y_0$	Probability
0 0	0 0	0 0	1/10
0 1	0 0	0 1	2/10
	0 1	0 0	
1 0	0 0	1 0	3/10
	0 1	0 1	
	1 0	0 0	
1 1	0 0	1 1	4/10
	0 1	1 0	
	1 0	0 1	
	1 1	0 0	

Table 1. Possible values for bits of x, y, and z preceding the target window $z(2)$ - no carry into addition window

$z_1\ z_0$	$x_1\ x_0$	$y_1\ y_0$	Probability
0 0	1 1	0 1	1/2
	1 0	1 0	
	0 1	1 1	
0 1	1 1	1 0	1/3
	1 0	1 1	
1 0	1 1	1 1	1/6

Table 2. Possible values for bits of x, y, and z preceding target window $z(2)$ - carry into addition window

Now suppose that there was a carry into the addition window. Six operand combinations produce this result. The possible values for (z_0, z_1) are found in table 2, along with the probability of occurrence and operands that produce those values. Again, z_0 and z_1 do not belong to the uniform distribution. Unlike the previous case, the values are now biased towards smaller values.

This can be understood more generally as follows. If integer addends X and Y, both in $[0, n-1]$, are added, the distribution of the result is the discrete convolution of the distributions of X and Y, and it takes on values in $[0, 2n-2]$. Further, if the distributions on X and Y are both uniform, then the probability of the sum increases linearly from its smallest value of $\frac{1}{n^2}$ at the value 0 to a maximum at $n-1$ and then decreases linearly to $\frac{1}{n^2}$ at the maximum value of the sum, $2n-1$. This may be understood by noting that many combinations of values x and y of the addends X and Y will give a sum of $n-1$, while only one pair of x and y give each of the values 0 and $2n-2$. Thus, if one looks at the variable z, the remainder of the sum modulo n, the value $n-1$ has the largest probability, and it only occurs when the carry is zero, i.e. $z < n$. Further, when $z \geq n$ (the carry is one) the largest value of the remainder ($n-2$, corresponding to the sum of $2n-2$) has the smallest probability, and the probability increases as the value of the remainder decreases, while the converse is true when the carry is one (see Figure 2). Thus, when addends are themselves the result of an addition, their distribution depends on the carry of the previous addition. In particular, after one addition of uniformly distributed values, the carry is slightly more likely to be zero than one. On adding two values, each the result of an addition of two uniformly distributed values, the result is more likely than not to produce a carry. Thus, while a zero carry is more likely when adding two uniformly distributed values, a sequence of zero carries for a sequence of additions is not always an optimal guess.

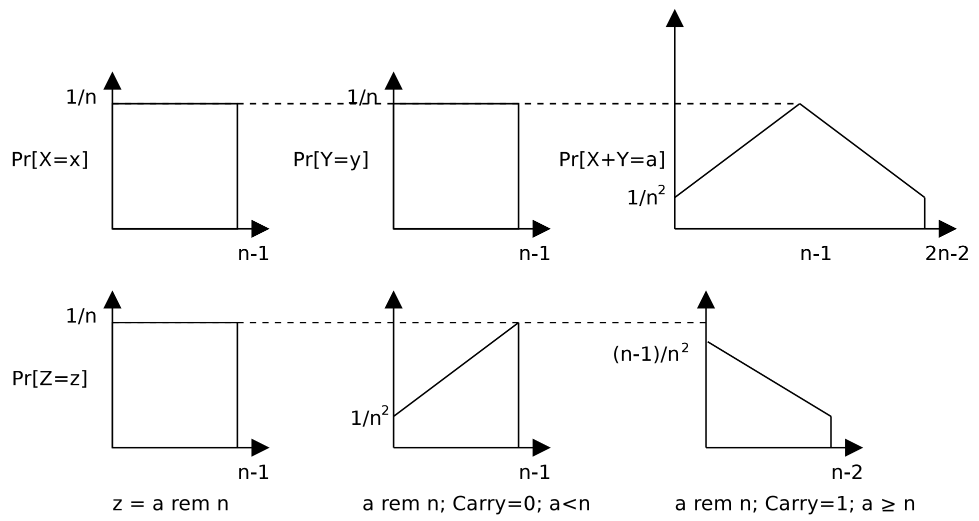


Fig. 2. Carries and addend distribution

3.2 Creating Approximations

Three aspects of the pseudo-linear approach are important.

Base Approximation The simplest approximation is formed by deriving the target window in terms of all the windows it depends on. All exclusive-or operations are preserved. Addition modulo 2^n operations are preserved as well, but the addends are incomplete. After each addition, bits outside the active windows are cleared (set to zero). For each unknown carry into a window, no carry is assumed.

Carry Patterns A carry pattern is a series of carry values, $c_i \in \{0, 1\}$, where each i denotes an approximated addition window that may have a carry into it. Stated differently, there is a c_i for every approximated addition window that does not start with the least significant bit of the word. One can construct multiple carry patterns that overlay the base approximation such that 1 is added to approximated addition window i if $c_i = 1$.

Let $C^j = (c_0 \dots c_{m-1})$ represent the carry pattern for m approximated addition windows. Then each base approximation overlaid with a distinct carry pattern, $\mathit{base} + C^j$, represents a distinct approximation. By applying several carry patterns to the same input/output pairs and obtaining several likely approximations, one can increase the probability of hitting upon the correct one, which in turn increases the bias.

The best carry pattern for a particular set of plaintexts depends on the input distribution. For example, the carry pattern for low Hamming-weight inputs will have fewer carries than for high Hamming-weight inputs. Note that this knowledge could potentially be used to find the Hamming weight of keys.

Offsets Another method of compensating for mispredicted carries is through the use of *offsets*. Recall that the central limit theorem says that probability distribution function of the sum of many random variables tends towards the gaussian function. We hence expect that the final values will be gaussian distributed about a mean, and that, in fact, strings close to the mean will be more likely to be correct than those away. While the expressions get complicated too quickly for a theoretical analysis, we see this is true in simulations, and it justifies the use of an integer *offset*$\in \{0, \cdots, 2^{w-1} - 1\}$ which is added to and subtracted from an approximated window value, *approx*. If every valid value of *offset* is tried and the number of observed correct guesses are plotted, the result is a bell-shaped curve with a peak at *offset* $= 0$, like the one shown in figure 3. The horizontal line shows the probability of guessing correctly at random (in this case, $\frac{1}{256}$).

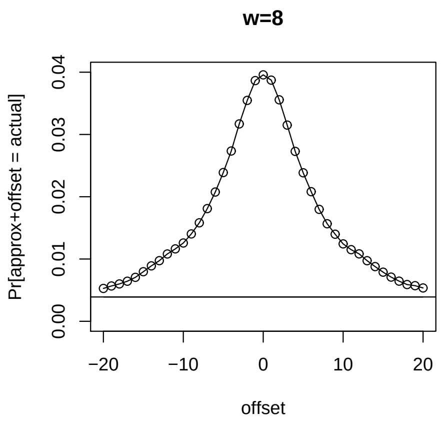


Fig. 3. An example: observed probability of correctly guessing window of ARX target over several offsets. Plot created with window isolation (section 3.4), the base approximation presented in section 6.1, and $w = 8$. Further details in section 6.

Computing Bias The improvements outlined above increase the probability of the approximation being correct, but also increase the probability of guessing correctly at random. That is, a random guess with cp different carry patterns will be correct with probability $\frac{cp}{2^w}$ instead of $\frac{1}{2^w}$ since each pattern represents a different approximation. Therefore, we need to adjust the bias calculation when using multiple carry patterns and/or offsets. Let T be the number of times the guess is correct, and P be the number of plaintext-ciphertext pairs. Let cp be the number of patterns used, and let f be the offset. Then the bias is calculated according to equation 1.

$$bias = T - \frac{cp(2f+1)}{2^w}P \tag{1}$$

3.3 Comparison to Linear Cryptanalysis

The main advantage of pseudo-linear cryptanalysis when compared with linear cryptanalysis is that it allows the adversary to create approximations that have higher bias than those that could be achieved by only considering approximations in the boolean vector space.

Another significant advantage is that pseudo-linear analysis leads to flexible approximations. Once a base approximation has been established, it can easily be modified through the use of offsets and carry patterns to accommodate different scenarios. For example, a change in the input distribution does not require a change to the base approximation, only a change to the carry pattern and/or offsets. It is also trivial to change the window size, since only the position of the least significant bit of the window is important. The most significant bit can be computed dynamically based on the desired w.

There are disadvantages that come with this method as well. The most notable is that approximations do not use a function of the key bits (such as the parity of some bits in linear cryptanalysis) but use the key bits themselves. As a result, pseudo-linear cryptanalysis has a higher attack complexity than linear cryptanalysis. This is because the two operations — exclusive-or and addition modulo 2^w — do not distribute. Because of this, one cannot combine all key bits in an approximation into a single function of the key. With linear cryptanalysis, on the other hand, the use of a single operation — exclusive-or — allows all key bits in an approximation to be combined into a single function of key bits. Pseudo-linear approximation requires the adversary to try all key bits used in the approximation. Even so, one can obtain attacks more efficient than brute force because pseudo-linear approximations enable us to reduce the number of key bits from those required by the cipher. One can approximate one of exclusive-or and addition modulo 2^w using the other, but this only works reasonably well when w is small, which is precisely when pseudo-linear approximations provide the weakest assistance. The larger w is, the less accurate approximating addition modulo 2^w with exclusive-or (or visa versa) becomes.

3.4 Implementation Methods

There are two implementations of pseudo-linear approximations over ARX functions that are demonstrated here.

1. Each window stored in its own word (window isolation)
2. All windows for a word stored in the same word (word isolation)

The first is useful for developing approximations, but the latter is much better in practice. This is because the first case provides a way to isolate windows that simplifies the debugging process. However, when windows are adjacent or overlap, there is carry information that is not used due to the high level of isolation. It also becomes possible for a single bit to take on multiple values in multiple windows, which is clearly not correct and will reduce the bias. In the second method, bits may no longer take on multiple values, and the attacker can take advantage of the extra carry information to obtain stronger biases.

The first approach also has another drawback. Because there is an addition performed for each addition window, the number of additions increases rapidly with each round. The second method uses at most twice the number of additions (1 for word addition, 1 for carry addition).

It is also essential that window position is preserved. When an addition is performed on windows that wrap around a word, it is important that the carry out of position $n-1$ is *not* propagated. Both implementation methods respect this. We stored windows in 64-bit words (regardless of isolation) in the following manner. The bits in a window were set to their correct value in the corresponding position, and all other bits set to 0. Let x(i) indicate the window beginning at position i in word x, and consider 64-bit words. With window isolation, $x(8)$ =0x2E would be represented as 00000000 00002E00, and all other windows would have their own distinct 64-bit word. With word isolation and two 16-bit windows, $x(8) =$ 0x5678 and $x(40) =$ 0x1234, the word would be represented as 00123400 00567800.

When using word isolation, the inactive bits may all be set to zero after each addition. Alternatively, they may remain after all operations, and be used throughout the cipher. The latter may be difficult to implement correctly depending on the method of deciding whether or not to add 1 when a carry is guessed. The first approach has been found to be more beneficial overall, although there were drops in bias at certain w values.

4 Truncated Differentials

Pseudo-linear methods are directly applicable to truncated differential attacks. A typical differential attack relies on differential characteristics, which are sequences of input and output differences over one or multiple rounds. A differential trail can be created by chaining characteristics to cover more rounds.

Pseudo-linear approximations can be used to discover differential trails – particularly truncated differential trails. Unlike a traditional differential which uses the entire state, a truncated differential works with a portion of the state. A common practice in differential calculation is to linearize the function (replace addition/subtraction modulo 2^n with exclusive-or). We propose to eliminate the linearization step and instead follow the differentials in windows.

Lemma 4. Let x and x' be two n-bit words, $x' \oplus x = \delta$, $x > x'$. Let w be the window size in bits. Then $\Pr[part(x - x', s, (s+w) \bmod n) = part(x, s, (s+w) \bmod n) \boxminus_w part(x', s, (s+w) \bmod n)] > \frac{1}{2}$

Proof. This follows directly from Corollary 2.

Figure 4 depicts a simple ARX function with a 1-bit difference in one of the words. The windows in the trace are highlighted in grey, with the difference labeled on top. In this example, the left word contains a 1-bit difference inside a window. There are no differences in the windows on the right. After one application

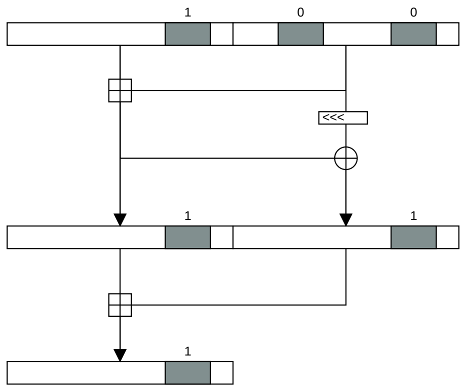


Fig. 4. Simple view of tracing differences

of addition, rotation, and exclusive-or, there is a 1-bit difference in the left window and in the right window, with some probability. After another addition, there is still a 1-bit difference, with some probability.

The differences can be traced from word to word throughout the function. How a difference in the input affects each window in the trace depends on the location of the differences relative to the windows. In particular, a difference occurring in less significant bits than a window has less impact as the distance between the start of the window and the difference increases. This is because the probability of the difference introduced by the carry function propagating to the window decreases as the distance between the difference and the window increases.

Consider the example in figure 4, but allow the possibility that there are differences outside the window which are possibly unknown. If the input difference of the left word is 1111_2, the difference of the addition will propagate differently than than if the input difference is 1000_2. If the window begins at the fourth least significant bit, then both result in a difference *inside* the window of 1, but the differences *before* the window produce different carry distributions, thereby affecting the difference propagation inside the window. If the window begins much farther away from other differences, then the impact will be smaller. For example, if the window began in position 32, and the difference outside the window only occurred in the least significant bit, then the probability of that difference affecting the window after one addition is much smaller because the probability of the carry function reaching that far is much smaller.

4.1 Difference in Addition

We now investigate how different operators affect the trail. First, consider how differences affect the approximation addition modulo 2^w.

Difference in one addend Let us start by considering two n-bit words, x and x', $x' = x \oplus 1$, and a third word y. When $z = x \boxplus y$ and $z' = x' \boxplus y$, there is a difference in the first addend, but there is no difference in the second addend. We can say something about the resulting difference $z \oplus z'$.

The simplest claim we can make is that the last bit of $z \oplus z'$ is the same as the last bit of $x \oplus x'$. When $x \oplus x'$ is even, this claim is straightforward. When $x \oplus x'$ is odd, this may be deduced as follows. Without loss of generality, let x be even and x' be odd, such that $x \oplus x'$ is odd. If y_0 (the least significant bit of y) is 0, then $z_0 = 0$ and $z'_0 = 1$. If $y_0 = 1$, then $z_0 = 1$ and $z'_0 = 0$. In both cases, $z \oplus z' = 1$.

More interesting to consider is how carries propagate. In particular, there are cases where strings of ones can be created or broken due carries. To see this, let us examine z'. Suppose $x \oplus x' = 1$ and that y is selected from the uniform distribution at random. Then $pr[z'_0 = 0] = \frac{1}{2}$. If $y_0 = 1$, then $carry_0 = 1$ and $z'_1 = \overline{z_1}$. Similarly, if $carry_1 = 1$ then $z'_2 = \overline{z_2}$. This continues until $carry_i = 0$. All of the other bits are the same in both x_0 and x'_0 for bits $i+1, \ldots, n-1$, and therefore produce the same values going forward. This can be generalized for a difference starting in any bit position.

If the input difference has a run of multiple ones (e.g. 3, 7, F, but not 1), some of the ones may turn to zeros in the output difference. For example, let $x = 1111_2$ and $x' = 1100_2$, giving $x \oplus x' = 11_2$. Let $y = 1101_2$.

$$\begin{array}{r} 1\,1\,1\,1 \\ \boxplus\, 1\,1\,0\,1 \\ \hline 1\,1\,0\,0 \end{array} \qquad \begin{array}{r} 1\,1\,0\,0 \\ \boxplus\, 1\,1\,0\,1 \\ \hline 1\,0\,0\,1 \end{array}$$

For $x \boxplus y$, carries result from positions 0, 1, and 2. For $x' \boxplus y$, carries result from position 2 only. The resulting difference characteristic here is $\Delta = 0011_2 \rightarrow \delta = 0101_2$, and the string of ones is broken.

In both addends It may be the case that both addends have difference patterns. For example, let $x' = x \oplus \Delta$, and $y' = y \oplus \Gamma$. Addition is performed as $z = x \boxplus y$, and $z' = x' \boxplus y'$, and $z \oplus z' = \delta$ for some value δ.

While the structure of the output difference is going to vary greatly by input differences, one can easily say something about the least significant bit of the difference pattern.

Consider some difference pattern `000XXX000` in Δ and Γ where `X` marks a possibly nonzero difference. Since there is no difference coming into the bits marked by `X`, the carry function into that region behaves the same way in both additions. Therefore we eliminate the zero regions from consideration and only consider `XXX`. One can say something about the least significant bit of `XXX` based on the least significant bit of the input difference patterns.

Lemma 5. *If both inputs have an even (odd) difference pattern, then the sum will have an even difference pattern. If one input has an odd difference pattern and the other an even difference pattern, then the sum will have an odd difference pattern.*

Proof. Let the function $lsb(d)$ return the least significant bit of the difference window. If the difference in both input windows is even, $lsb(x) = lsb(x')$ and $lsb(y) = lsb(y')$. Hence,

$$lsb(z) \oplus lsb(z') = (lsb(x) \oplus lsb(y)) \oplus (lsb(x') \oplus lsb(y')) = (lsb(x) \oplus lsb(x')) \oplus (lsb(y) \oplus lsb(y')) = 0$$

Now suppose the difference in both input windows is odd, so $lsb(x) = \overline{lsb(x')}$ and $lsb(y) = \overline{lsb(y')}$. Hence,

$$lsb(z) \oplus lsb(z') = (lsb(x) \oplus lsb(y)) \oplus (lsb(x') \oplus lsb(y')) = (lsb(x) \oplus lsb(x')) \oplus (lsb(y) \oplus lsb(y')) = 1 \oplus 1 = 0$$

Finally, suppose one addend has an odd difference pattern and the other an even difference pattern. Without loss of generality suppose $lsb(x) = lsb(x')$ and $lsb(y) = \overline{lsb(y')}$. Hence:

$$lsb(z) \oplus lsb(z') = (lsb(x) \oplus lsb(y)) \oplus (lsb(x') \oplus lsb(y')) = (lsb(x) \oplus lsb(x')) \oplus (lsb(y) \oplus lsb(y')) = 0 \oplus 1 = 1$$

4.2 Difference in Exclusive-Or

When a difference occurs in only one of the operands, that difference is preserved by exclusive-or. Consider n-bit words a, $a' = a \oplus \Delta$, and b, and let $c = a \oplus b$ and $c' = a' \oplus b$. Then $a' \oplus b = a \oplus \Delta \oplus b \rightarrow c' \oplus c = \Delta$.

4.3 Differences Over Round Functions

The simplest cases to analyze are those with low Hamming weight differences. Consider $x' = x \oplus 2^i, i \in \{0, \ldots, n-1\}$.

The difference characteristics between rounds can be traced by following the effect of the carry. The input difference will affect the differential in varying ways depending on its position. For example, cryptanalysts have placed the difference in the most significant bit of a word, since this minimizes the effect of the carry. This is because during an addition operation, no combination of addends will produce a carry out of that bit due to modular addition. In other words, addition of the most significant bit of a word behaves like exclusive-or in addition modulo 2^n. However, this practice becomes ineffective after some time as the rotation will move the difference to a position where the carry will matter. That is, a difference that was outside the target and thus was ignored should eventually affect the target window as the trail covers more rounds because the rotation will move the difference bits into a position that affects the window. Note that this may not be true when rotation amounts are poorly selected.

4.4 Windows and Truncated Differentials

A differential trail, truncated or otherwise, typically covers $r-1$ of r rounds. The adversary chooses inputs P and $P \oplus \Delta$, and receives their encryptions C and C'. Next, the adversary guesses portions of the key that are needed to compute the target window (where the target difference occurs), and backs up one round to obtain T and T'. If $T \oplus T' = \delta$, where δ is the target difference, then increment a counter for those key bit values. At the end, the guessed key bits that produced $T \oplus T' = \delta$ at the expected rate are the most likely candidates.

Windowing is used differently for differentials than for pseudo-linear approximations. This is because higher accuracy is required when working with differentials. Specifically, target windows should be formed such that no subtraction is left to chance. The word should be covered from the least significant bit to the highest position across all relevant windows so that all borrows are known.

One strategy is to start with small windows, say $w = 4$. Once all the bits involved with that window size have been found, increase w. The window position does not change, but more of the unknowns are slowly added to the windows. This proceeds until all n bits of a key word are active and the entire word has been guessed, *or* the correct key is no longer distinguishable from incorrect keys. This can happen if all the new key bits guessed do not change the probability of resulting in the target difference. Note that the difference in a word is the difference over the entire n bit word, where inactive bits are zero.

5 Evaluating ARX Functions

Diffusion in ARX is achieved primarily through rotation, and secondarily by the carry function of the addition. There is little work which describes how to construct a good round function, and how to determine how good an existing round function is. *Branch number* is a metric currently in use, and was used in the design strategy for the Advanced Encryption Standard (AES) [7].

5.1 Branch Numbers

The differential branch number and linear branch number metrics are measures of diffusion that apply to any mapping. Here, their use is restricted to the context of key-alternating round functions. Branch numbers provide lower bounds for the amount of diffusion in a round function, allowing simpler analysis of the weight (or probability) of a trail over several rounds. The metrics have been defined for any partition [6], as well as a specific type of partition [7]. In [7], branch numbers are defined as part of an m-bit partition of the state, consisting of *bundles*. Bundles are adjacent and do not overlap, such that the state is divided into n_b bundles and $n_b m$ is the size of the state in bits.

A bundle is called *active* when it is part of a differential or linear trail. That is, when a difference or selection pattern is non-zero for that bundle in a differential or linear trail, respectively. The *bundle weight* of a state is defined as the number of active bundles in that state. The bundle weight of state a is written $w_b(a)$.

Branch number metrics are defined differently for differential and linear trails [7]. As the number of rounds increases, the branch number becomes more complex and difficult to reason about. However, it has been shown that properties of round functions can be derived by looking at two-round trails, and viewing $2r$ rounds as a sequence of r two-round trails[7]. Therefore, only two-round trails will be discussed here.

For input vectors a and b, the differential branch number for round transformation ϕ is defined as [7]:

$$B_d(\phi) = \min_{a, b \neq a} \{w_b(a \oplus b) + w_b(\phi(a) \oplus \phi(b))\} \tag{2}$$

Generally speaking, this is the minimum bundle weight of the input difference to ϕ added to that of the output difference of ϕ. This is meaningful because it provides a lower bound on the propagation of the input difference pattern to the output of ϕ, and thus illustrates a lower bound on the diffusion of a single bundle or set of bundles.

Linear branch number is similar. For an input selection pattern α, output selection pattern β, correlation function $C(.)$, and input x, the linear branch number of a round transformation ϕ is defined as [7]:

$$B_l(\phi) = \min_{\alpha,\beta,C(\alpha^T x,\beta^T \phi(x))\neq 0} \{w_b(\alpha) + w_b(\beta)\} \tag{3}$$

Generally speaking, this is the minimum of the sum of bundle weights for the inputs and outputs of ϕ over all selection masks, where the active input and output to the ϕ function are correlated.

Branch Numbers in ARX This metric works well for states that are comprised of a sufficient number of bundles. However, diffusion in ARX round functions comprised of few large bundles (e.g., four 32-bit or 64-bit words) may be difficult to accurately capture with this metric.

To see why, consider the scenario in figure 5, where two round transformations have approximately the same amount of diffusion, but one round function is an SPN that operates on bytes (figure 5(a)) and the other is an ARX function using 32-bit words (figure 5(b)). In both round functions, the same bits are part of a linear or differential trail. Portions of the state (byte or word) that contain at least one bit of the trail are called active, and are colored grey in figure 5. When operations are performed in 32-bit words, each word covers four bytes. As a result, all of `state 1` appears active in 5(b), whereas only $\frac{3}{8}$ of `state 1` is active in 5(a). Similarly, $\frac{5}{8}$ of `state 2` is active in figure 5(a), while again being completely active in figure 5(b). In this example, it appears that the diffusion is far better in 5(b), when it really is not. A state composed of few large words can produce branch numbers that are misleading.

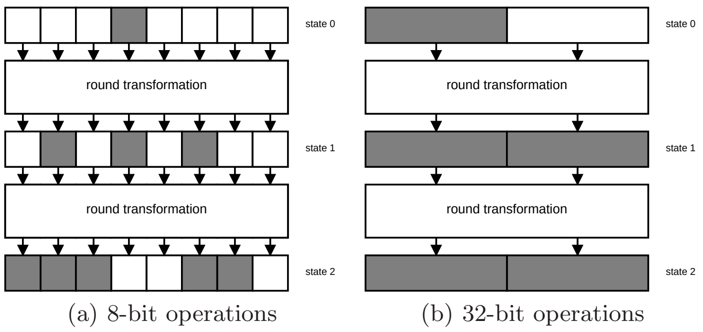


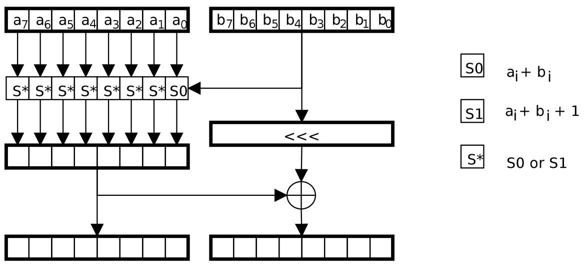
Fig. 5. Trail in two transformations with approximately the same amount of diffusion

We propose two approaches to more accurately measure diffusion in this type of round function. The first proposal is based on the idea of breaking large words into smaller ones, and modeling addition as data dependent substitution boxes. The second approach leverages concepts of pseudo-linear analysis.

S-Box Representation Addition modulo 2^n can be represented as a series of smaller data-dependent substitution boxes, as shown in figure 6. There are two kinds of S-boxes:

S_0**:** $a_i = a_i + b_i$
S_1**:** $a_i = a_i + b_i + 1$

Each bundle may be thought of as a window. Starting with the least significant window, apply S_0 to a_0. Note that the result is also dependent on the bundle b_0. Moving from the least significant bundle towards the most significant bundle, use $S_0(a_i)$ when there is not a carry out of the previous bundle addition. When there is a carry out of the previous bundle addition, $S_1(a_i)$ is used. This gives the same result as the original function.



a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0
b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0
S* S* S* S* S* S* S* S0
<<<
S0 $a_i + b_i$
S1 $a_i + b_i + 1$
S* S0 or S1

Fig. 6. S-box representation of an ARX function

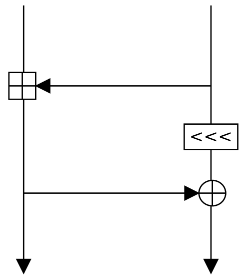


Fig. 7. Threefish mix function; rotation varies by state and position

A bundle is not considered active in a differential trail unless there is a non-zero difference in the input to the corresponding S-box. That is, if there is a non-zero difference in the input of an S-box step, but one input is to S_0 and the other is to S_1, the bundle will *not* be considered active. The rationale for this is that branch number provides a lower bound, so only the minimum number of S-boxes affected is considered here.

Alternate Metric Pseudo-linear analysis lends itself to another possibility for application of the branch number metrics. If the restriction that bundles be non-overlapping adjacent portions of the state were removed, then bundles would strongly resemble windows in a pseudo-linear system. In fact, when this restriction is removed, pseudo-linear approximations are directly applicable to the bundles to trace dependencies and influence. It is this idea that motivates an alternate metric for diffusion in ARX rounds, called *effective branch number*.

The very first step is to determine the proper window (bundle) size. In pseudo-linear cryptanalysis, the bounds on window size w are defined as $1 \leq w < n$.

Just as the original branch number metrics used bundle weight, these variants use *effective bundle weight*, denoted ew_b. Effective bundle weight is calculated as the ceiling of the total number of active bits divided by the bundle size.

Example 1. Consider a round function and approximation with five active input windows, but all five windows start one bit off from another. That is, we have windows starting in positions $i, i+1, i+2, i+3$, and $i+4$. The only bits that are not contained in multiple windows are the first bit of window $x(i)$ and the last bit of window $x(i+4)$. The bit range covered by these windows is $[i \ldots i+4+w)$. The number of state bits in this range is $4+w$.

If $w = 2$, then the effective bundle weight is $\lceil \frac{6}{2} \rceil = 3$. If $w = 4$, then the effective bundle weight is $\lceil \frac{8}{4} \rceil = 2$. If $w = 8$, then the effective bundle weight is $\lceil \frac{12}{8} \rceil = 2$.

The definitions for linear and differential effective branch number are simple extensions of the branch number metrics from [7]. They only differ in that effective bundle weight is used in place of bundle weight. We denote the differential effective branch number as EB_d and pseudo-linear effective branch number as EB_{pl}.

Over two-round trails, the differential effective branch number of a transformation ϕ is derived by equation 4. Recall from equation 2 that a and b are input vectors to the first round, and $\phi(a) \oplus \phi(b)$ is the difference at the input to the second round.

$$EB_d(\phi) = \min_{a, b \neq a, w} \{ew_b(a \oplus b) + ew_b(\phi(a) \oplus \phi(b))\} \tag{4}$$

Similarly, the pseudo-linear effective branch number of a transformation ϕ is derived by equation 5. The selection mask of the input to the first round is α, the selection mask at the input to the second round is β, and $C(.)$ is a correlation function over the masks and an input vector, x. Note that the linear effective branch number is the pseudo-linear effective branch number where $w = 1$.

$$EB_{pl}(\phi) = \min_{\alpha,\beta,C(\alpha^T x,\beta^T \phi(x))\neq 0,w} \{ew_b(\alpha) + ew_b(\beta)\} \tag{5}$$

The matter of selecting an appropriate w has yet to be addressed. The minimum useful window size is that which allows a round function to be distinguished from a random permutation. This in turn depends on the number of additions required to compute a target window, denoted by $n_{\boxplus}$. If windows do not overlap, then the effective branch number should be equivalent to the branch number calculations of [7]. To satisfy this requirement, w should divide the number of bits in the state. Thus, for state width that is a power of 2, let $w = 2^{\lfloor lg(n_{\boxplus})\rfloor}$. Note that only the additions occurring in the key-independent portion of a round function should be used to derive the appropriate w.

6 Application: Threefish

Threefish[8] is the tweakable[17] block cipher inside the compression function of SHA-3 finalist Skein[10]. Two variants are discussed here: Threefish-256 and Threefish-512. The state and key are comprised of 64-bit words, where there are B bits in the state and the key of Threefish-B. The cipher consists of 72 rounds, where each round contains two main steps: mix and permute.

The mix function, shown in figure 7, is an ARX function where addition is performed modulo 2^{64} and the rotation depends on the mix inputs and round. Each mix is numbered by 0, 1, 2, or 3, where each is computed as $mix_0(x_0, x_1)$, $mix_1(x_2, x_3)$, $mix_2(x_4, x_5)$, and $mix_3(x_6, x_7)$, in Threefish-512. Threefish-256 requires only $mix_0(x_0, x_1)$ and $mix_1(x_2, x_3)$.

The rotational values for each mix in each round can be found in tables 3 and 4.

Round mod 8	**Mix 0**	**Mix 1**
0	5	56
1	36	28
2	13	46
3	58	44
4	26	20
5	53	35
6	11	42
7	59	50

Table 3. Threefish-256 Rotation Constants (round 2)

Round mod 8	**Mix 0**	**Mix 1**	**Mix 2**	**Mix 3**
0	46	36	19	37
1	33	27	14	42
2	17	49	36	39
3	44	9	54	56
4	39	30	34	24
5	13	50	10	17
6	25	29	39	43
7	8	35	56	22

Table 4. Threefish-512 Rotation Constants (round 2)

The permutation step of the round function is straightforward. Each 64-bit word is mapped to another position according to the mapping, π, given in table 6. That is, $x_i \leftarrow x_{\pi(i)}$.

i	0	1	2	3
$\pi(i)$	0	3	2	1

Table 5. Threefish-256 Permutation, π

i	0	1	2	3	4	5	6	7
$\pi(i)$	2	1	4	7	6	5	0	3

Table 6. Threefish-512 Permutation, π

Every four rounds, a round key is injected via addition modulo 2^{64}. Threefish has a simple key schedule that is quick to compute and has a small footprint. The secret key is 256 bits for Threefish-256 and 512 bits for Threefish-512, organized as four and eight 64-bit words, respectively. Let N denote the number of 64-bit words in the key and state. The exclusive-or of all N words and a constant form an additional key word, the parity word $N + 1$. In addition to the $N + 1$ key words, there are three tweak words. Two tweak words, t_0 and t_1, are given as input and the third tweak word, t_2, is the result of $t_0 \oplus t_1$. In the hash function, these words store information about what has been done so far (bits processed, level in the hash tree, etc.). No structure of the tweak words is assumed when the cipher is considered on its own.

The Threefish key schedule is given by equations 6-9, where *constant* is a constant that was changed between the first [8] and final [10] submissions to NIST. For the purpose of this work, the value of the constant is not relevant[1]. Therefore, it is left as a variable to include all submitted versions of the algorithm. RK_i is round key $i \in \{0, \ldots, 18\}$, where RK_0 is the whitening key. Each round key is derived from the $N+1$ key words (denoted k_j, $j \in \{0, \ldots, 8\}$) and three tweak words (t_0, t_1, t_2).

$$k_N = constant \oplus \bigoplus_{i=0}^{N-1} k_i \tag{6}$$

$$t_2 = t_0 \oplus t_1 \tag{7}$$

Round key i for Threefish-256 is derived by

$$RK_i = k_{i \bmod 5} || k_{(i+1) \bmod 5} + t_{i \bmod 3} || k_{(i+2) \bmod 5} + t_{(i+1) \bmod 3} || k_{(i+3) \bmod 5} + i \tag{8}$$

Similarly, the key schedule for Threefish-512 is expressed by

$$\begin{aligned} RK_i = {} & k_{i \bmod 9} || k_{(i+1) \bmod 9} || k_{(i+2) \bmod 9} || k_{(i+3) \bmod 9} || k_{(i+4) \bmod 9} \\ & || k_{(i+5) \bmod 9} + t_{i \bmod 3} || k_{(i+6) \bmod 9} + t_{(i+1) \bmod 3} || k_{(i+7) \bmod 9} + i \end{aligned} \tag{9}$$

6.1 Pseudo-Linear Attacks on Threefish-256

Approximation Over Modified 8 Rounds The application of pseudo-linear attacks to a modified version of Threefish-256 is presented in this section. The modified version has a fixed tweak schedule of zero, is reduced to 8 rounds, and does not include the injection of a whitening key. This approximation covers the modified cipher from the key injection of round 4 to the key injection at round 8, inclusive.

Let x_i denote the i^{th} word of the plaintext and x_i^j the i^{th} word after round j. Let $x_i^j(m)$ denote the window of word x_i^j with its least significant bit in position m. That is, $x_i^j(m) = part(x_i^j, m, (m+w) \bmod 64)$.

When a single word contains multiple active windows, $a_0 \ldots a_m$, let $x_i^j(a_0, \ldots, a_m)$ be the same as $x_i^j(a_0), \ldots, x_i^j(a_m)$. $K_i(m)$ denotes the window starting at position m of the i^{th} word of the key schedule.

The approximation for $x_0^8(0)$ is presented in figure 8. The active addition windows are listed to the upper left of each $\boxplus$ symbol. $rotl(r)$ represents a left rotation by r bits of a 64-bit string.

Computations before the fourth round key addition can be performed by following the cipher operations, and are not included here. The number of key bits needed in this approximation depends on the window size. For example, $w = 3$ requires 58 key bits, $w = 4$ requires 75 key bits, and $w = 5$ requires 92 key bits.

Empirical results for bias were obtained with varying window sizes, using 20,000,000 pseudorandomly generated (data, key) pairs and tweak schedule fixed to 0. This methodology is based on the one used in the original Skein submission to the SHA-3 competition [8], where differential biases were computed using 20,000,000 pseudorandom (data, key, tweak) tuples.

As mentioned in section 3.4, there were two methods of implementation used – the less efficient and less accurate but easier to debug window isolation, and the more accurate and more efficient but more difficult to debug word isolation.

For simplicity, a carry pattern C^i is represented as an array C^i_{jk}, where j is the round number and k is the word number. In these experiments, if a word k of round j contained multiple windows, the same carry guess C^i_{jk} was applied to all of those windows.

[1] This is in contrast to rotational cryptanalysis, which is a differential technique that relies the same properties of the carry functions as this work. The original constant of repeated bytes only had two values under all rotations, and was changed to prevent rotational cryptanalysis of Threefish.

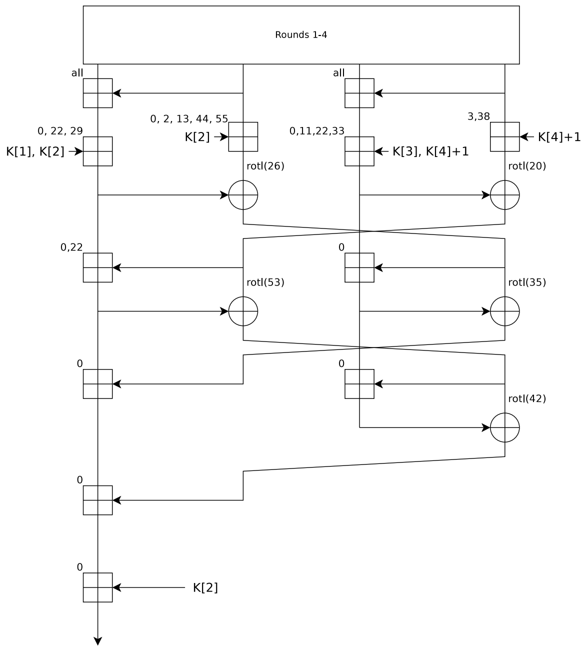


Fig. 8. Approximation for $x_0^8(0)$, without whitening

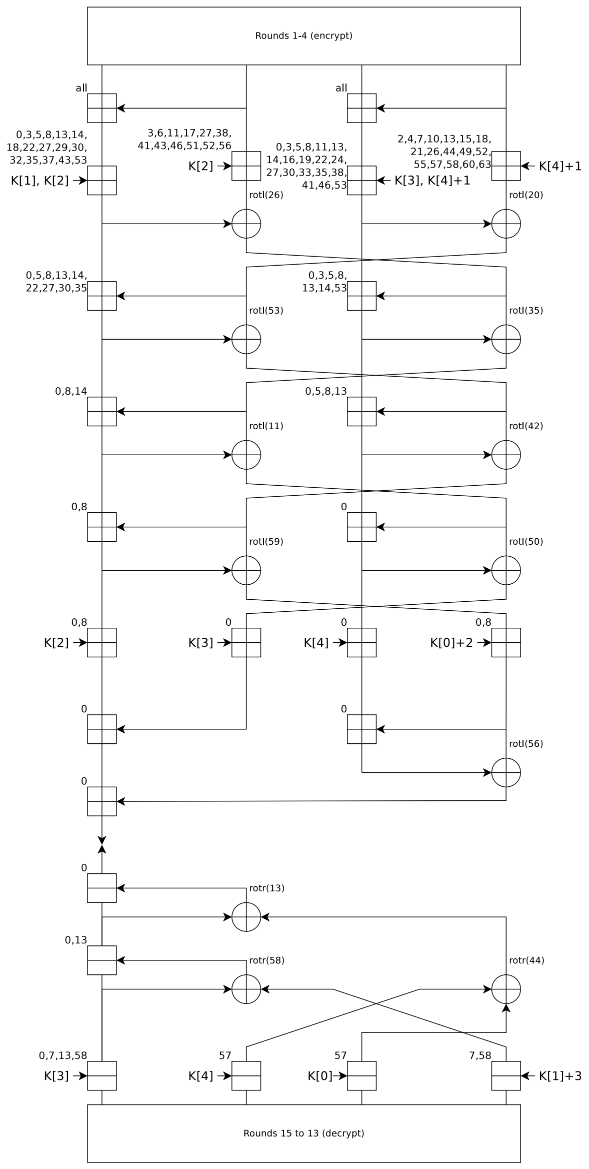


Fig. 9. Approximation for 12 rounds, meeting at $x_0^{10}(0)$, without whitening

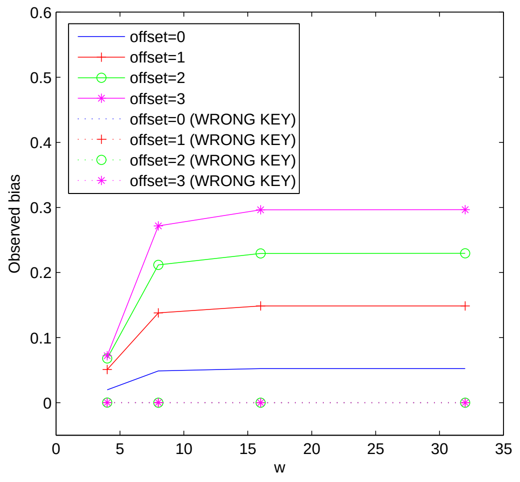


Fig. 10. Base approximation with window isolation, $x_0^8(0)$

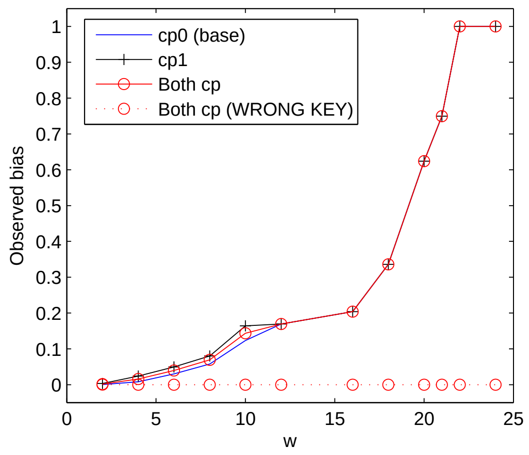


Fig. 11. $x_0^8(0)$ approximation with word isolation and different carry patterns (cp)

Figure 10 demonstrates the effect of offsets on an approximation, with both the correct and incorrect key values, where observed bias is computed according to equation 1. Offsets increase the probability of correctness for wrong values considerably because they also result in larger solution sets. On the other hand, correct values continue to have a higher bias on average. The same is true when multiple carry patterns are used, and the resulting solution set consists of all the solutions resulting from the carry patterns used. Figure 10 also shows that there is a maximum bias that can be achieved with window isolation, where the maximum probability of correctness is strictly smaller than 1.

Figure 11 presents some results obtained using word isolation and *offset*=0. It demonstrates that word isolation provides better results, as expected, and that the correctness probability grows to 1 as w increases. This causes the bias to approach 1, since the probability of correctness for a random guess drops closer to zero as w increases. Finally, it shows how different carry patterns can affect the approximation. The two carry patterns used here were:

$$C_j^0 k = 0,\ C_j^1 k = \begin{cases} 0 & \text{if } j \text{ is even} \\ 1 & \text{if } j \text{ is odd} \end{cases}$$

When a large enough number of key bits is to be guessed, the bias values for different carry patterns converge (here, this happens at $w = 12$). If multiple carry patterns are used together, as they were here, the resulting bias may be smaller than the bias obtained by using a single, stronger carry pattern.

Approximation Over Modified 12 Rounds The application of pseudo-linear attacks to a modified version of Threefish-256 is presented in this section. The modified version has a fixed tweak schedule of zero, is reduced to 12 rounds, and does not include the injection of a whitening key. This approximation covers the modified cipher from the key injection of round 4 to the key injection at round 12, inclusive.

Figure 12 contains results from the 12 round approximation with carry pattern C^1 and different offsets using word isolation. It shows that the correct key bits are indistinguishable for $w < 5$. With $w = 5$, 232 of 256 bits of key are involved in the approximation.

To demonstrate key recovery using pseudo-linear approximations, a smaller attack was performed with the 8 round approximation for $x_0^8(0)$. Again, this gives an attack on 11 rounds of Threefish-256, since there is not another key injection until round 12.

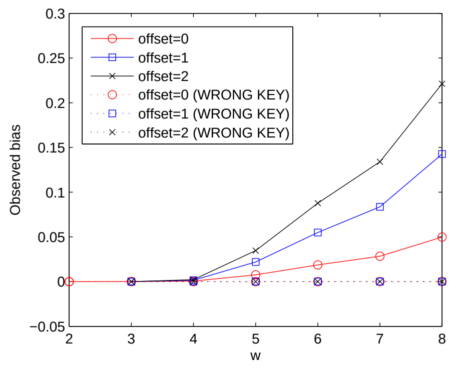


Fig. 12. Bias Threefish-256 12 round approximation

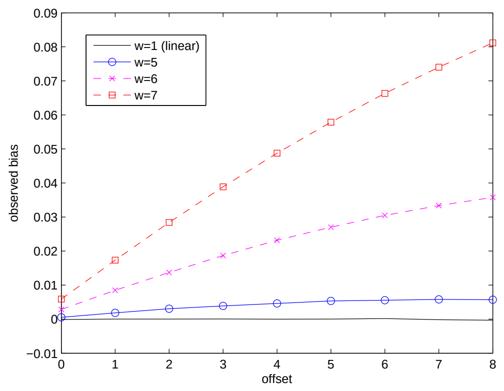


Fig. 13. Observed bias for Threefish-512 8 round distinguisher

If $w = 3$, then the attack on 11 rounds requires 58 bits of key (as compared to 256 bits for a brute force attack) to be guessed. Without expending too much energy on computation, in order to observe the kinds of errors that might result, we attempted to determine 18 key bits assuming that the other 40 were known.

Because the carry function is difficult to determine with incomplete knowledge of the operands, the correct value of the key is not always among the top few candidates. In these experiments, the goal was to eliminate 90% of the possible key values, with the correct value in the remaining 10%. We ran 500 independent key recovery attacks, each with a pseudorandomly generated key and 10,000 pseudorandomly generated plaintexts. The results are summarized in table 7, where we note when the correct key value has the maximum bias, as well as when the correct value is in the top 10% of key estimates ranked by bias.

	offset=0	offset=1	offset=2	offset=3
correct	11%	12.2%	14.4%	7.2%
in top 10%	92.6%	97.4%	96.8%	92.2%

Table 7. 500 attacks, 10,000 pairs

Key Recovery These approximations can be used for key recovery with known plaintext and ciphertext. Because there is no key injection in rounds 15 through 13, the output of round 15 can be decrypted to derive the output of round 12 correctly. The input up to the fourth round key injection can also be computed precisely because the modified cipher does not include the whitening key injection. In this scenario, the 12 round approximation could be used to determine some of the key bits from 15 rounds.

These results show that it is possible to eliminate many of the incorrect values while keeping the correct key bits with high probability, using only a portion of the key. While a larger value of the offset always helps in the task of approximating the output of the cipher, it does not always help in estimating key bits. This is because a larger offset is not necessarily more accurate, it simply encompasses a larger number of values. This will be true for incorrect key bits as well as correct key bits. For this reason, a larger window does not always help to effectively distinguish among correct and incorrect key bits. This can be seen in table 7 when the offset increases from 2 to 3. At *offset*=3, the number of values is so large that it becomes less effective in distinguishing the correct key bits.

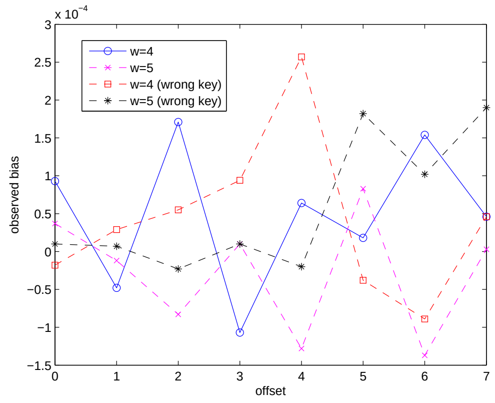


Fig. 14. Observed bias for 12 round Threefish-512 distinguisher, random messages

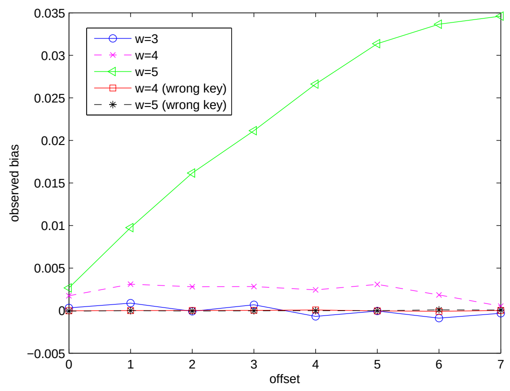


Fig. 15. Observed bias for 12 round Threefish-512 distinguisher, low Hamming weight messages

6.2 Pseudo-Linear Attacks on Threefish-512

In the previous two attacks, a modified Threefish-256 was used where the whitening step was eliminated. In the following approximations, the only modification made is the number of rounds. That is, the first key injection considered is the key whitening.

Approximation Over Modified 8 Rounds An 8-round approximation was constructed with a meet-in-the-middle approach. It meets at word 1 at the end of the fourth round. The addition (subtraction) windows for this distinguisher are presented in Appendix A, table 13. There are no subtraction windows in the fifth round because no subtractions are needed to obtain $x_1^4(0)$.

Empirical results for this distinguisher are presented in figure 13. These results were obtained using 20,000,000 randomly chosen $(plaintext, key)$ pairs with the carry pattern where carries are assumed every other round. It is clear from the plot that eight rounds of Threefish-512 are distinguishable for $w \geq 5$.

The number of key bits required by this distinguisher for window sizes $w = 5$, $w = 6$, and $w = 7$ are 310, 355, and 391, respectively. Recall that for a distinguisher from Threefish-256, eight rounds with $w = 5$ (seen in the modified 12 round distinguisher, section 6.1) required 232 of the 256 bits, or about 90%. In this eight round distinguisher for Threefish-512, a window size of 5 requires only about 60% of the 512 key bits to be guessed. The principal reason for this is that with the increased state size, diffusion between words is slower. Specifically, it takes more applications of the permute function, and therefore more rounds, to mix all eight state words.

12 Round Approximation A 12-round approximation can be obtained in a similar fashion to the 8 round one. In particular, one was constructed that meets in the middle at word 1 of the seventh round. The addition (subtraction) windows for this distinguisher are presented in Appendix A, table 14.

The key bits that need to be guessed for $w = 4$ and $w = 5$ are 509 and 511, respectively. Beyond that, all 512 bits need to be guessed for this distinguisher.

Empirical results are presented in figure 14 and 15 for two different plaintext selection criteria. In figure 14, 20,000,000 $(plaintext, key)$ pairs are chosen at random. The alternating carry pattern was used. In figure 15, key is still chosen at random, but the plaintexts are chosen such that they have a hamming weight of 1. For each randomly selected key, 512 plaintexts are used. Carries are assumed in rounds 6 and 8.

Figure 14 shows that one cannot effectively distinguish the correct key from incorrect keys when $w \leq 5$ and plaintexts are selected at random. When $w > 5$, all 512 bits need to be guessed.

However, when the model changes to allow the adversary to only choose low Hamming weight messages, the results improve. Figure 15 shows that when the message has a Hamming weight of 1, the correct key can be distinguished from a random permutation with $w = 4$ and offset ≤ 6. However, it cannot be distinguished from a random permutation for $w \leq 3$, so 509 bits is the minimum that must be guessed for this distinguisher.

6.3 Truncated Differentials for Threefish-512

In this section, we construct a truncated differential attack on four rounds of Threefish-512. This is a single-key chosen-plaintext attack with a pre-determined difference between pairs of plaintexts. The focus is on differences in window bits. Differences in bits outside the window are ignored.

Let the input be $x'_0 = x_0 \oplus 1$, $x'_i = x_i$ for $i \in \{1 \ldots 7\}$, and the target window be $x^3_3(55)$. The size of the window will begin small, such as $w = 4$, and then increase as the adversary gains more information.

The target window location is not arbitrary. Recall from section 4.4 that we need to remove uncertainty introduced by the carry by working with the least significant window for all subtractions. Since there is a right rotation by 9 after the key subtraction, we have $(0 - 9) \mod 64 = 55$, so the target window for x^3_3 starts at position 55.

To see how the target window is affected by the input difference, it is helpful to observe how the difference propagates through the trail for x^3_3. The first three rounds of Threefish-512 are shown in figure 16. The trail is marked with thick arrows.

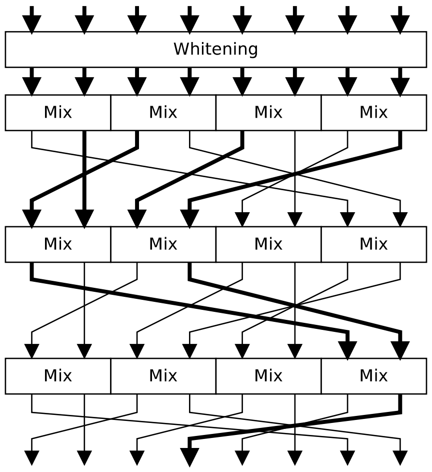


Fig. 16. Three rounds of Threefish-512

First Round The first round begins with the key whitening. The same key will be added to both plaintexts, x and x', which are equal except in the least significant bit of word 0. Words 1 through 7 have no difference before and after the key injection.

In word 0, after key injection, the difference will consist of a string of $n - t$ zeros followed by t ones, $1 \leq t \leq n$. The reason for this is as follows (recall section 4.1). Without loss of generality, let x_0 be even and x'_0 be odd. Then $x_0 \boxplus k_0 = x^*_0$ will not cause a carry out of position 0 regardless of the value of k. If the least significant bit of k_0 (denote this as $k_0[0]$) is 0, then $x'^*_0[0] = 1$. If $k_0[0] = 1$, then $x'^*_0[0] = 0$. Assuming that k is selected from the uniform distribution at random, then $pr[x^*_0[0] = 0] = \frac{1}{2}$. If $k_0[0] = 1$, then $carry_0 = 1$ and $x'^*_0[1] = \overline{x^*_0[1]}$. This continues until $carry_{t-1} = 0$. Bits in position $t, \ldots, n-1$, are identical in x_0 and x'_0 and thus the run of ones ends at after t bits.

A short list of possible difference values is provided in table 8. The approximated probabilities were obtained empirically over 20,000,000 randomly selected (x, key) pairs, where x' was derived from x.

Word	Input difference ($\varDelta_{x_0}$)	Output difference (δ)	$Pr[\delta \vert \varDelta_{x_0}]$ (approx)
x_0^*	1	1	2^{-1}
		3	2^{-2}
		7	2^{-3}
		F	2^{-4}
		1F	2^{-5}

Table 8. Key whitening difference probabilities

The first key injection is followed by the mix and permute steps. Recall that Threefish-512 computes four mix functions in parallel, where each mix function consists of an addition, word rotation, and exclusive-or. After mix function $i \in \{0, 1, 2, 3\}$ has been applied to words $2i$ and $2i + 1$, the words of the state are permuted to facilitate diffusion. There is no difference between words 1 in the two texts before the addition of the mix, but the words 0 differ by a string of ones. The rotation is performed on words 1, but there is no difference between them and thus there is still a zero difference after rotation. The final step of the mix is the exclusive-or of the rotated words 1 and result of the addition modulo 2^n. As a result, the words 0 still differ by a string of ones after the mix, and this string of ones is also the difference between the words 1 after the mix because these words differ from the corresponding words 0 only in that they are xored with the same string.

Permutation does not change the value of a difference within a word, only the word that it appears in. In particular, the string of ones forming the difference in word 0 after the mix step moves to word 6 after permutation, and the zero difference in word 2 moves to word 0 (see table 6). Word 1 maps to itself during this step, so at the end of round 1, the only non-zero difference bits are in words 1 and 6.

Some of the higher probability characteristics are summed up in table 9. These probabilities are dependent on the input difference to the first round mix function. That is, they are conditional probabilities where the result depends on the difference after the whitening step.

Second Round The incoming differences are in x_6^1 and x_1^1. x_6^1 will affect output windows x_4^2 because word 6 will be permuted to word 4 after the addition, and x_3^2 because word 7 will be swapped with word 3. x_1^1 will affect x_6^2 as in the first round, because the zeroth word is permuted to the sixth position, and the first word is not affected by the permutation, and x_1^2. However, x_1^2, x_3^2, and x_4^2 are not in the dependency path for x_3^3, as can be seen by following the words x_3^3 is derived from, and can be ignored for this differential. This leaves only x_6^2.

x_6^2 (derived from x_0^1 and x_1^1) is interesting because the format of the words is different from those we've seen before. In previous rounds, the differences in the left words have been strings of ones, but neither of the texts, at this part of the cipher, themselves contained the difference runs in the corresponding words. For example, a difference of 3 would be achieved by having the least significant bits 01_2 in one word and 10_2 in the other. In the input word x_1^1, one of the words contains the difference. That is, one word contains $\varDelta_{x_1^1}$ in the least significant bits, while the other contains zeros. For a difference of 3, this would be 11_2 in one word and 00_2 in the other. Section 4.1 showed how the run of ones in the difference pattern can break in this scenario. Because of this, differences that are not runs of ones are possible, such as `0x05` and `0x0B`.

Third Round Only one word in the trail has a difference into round 3: x_6^2. x_6^2 affects x_3^3.

Again, a difference in the right input creates more interesting transitions. The output difference probabilities for x_3^3 do not all contain perfect powers of 2 in the denominator. A subset of the possible transitions is shown in table 10.

We have showed the effect of the input difference throughout three rounds of a trail. This information can be used to construct a truncated differential attack on four rounds.

The trail covers the first three rounds – from the key whitening to the output of the third round. This is reflected in figure 16, where thicker lines indicate part of the trail. To perform the attack, the adversary guesses the round key of the target window. In this case the adversary guesses part of k_1 and

Word	Input difference ($\Delta_{x_0^*}$)	Output difference (δ)	$Pr[\delta\|\Delta]$ (approx)
x_6^1 and x_1^1	1	1	2^{-1}
		3	2^{-2}
		7	2^{-3}
		F	2^{-4}
		1F	2^{-5}
	3	1	2^{-1}
		3	2^{-2}
		7	2^{-3}
		F	2^{-4}
		1F	2^{-5}
	7	1	2^{-1}
		3	2^{-2}
		7	2^{-3}
		F	2^{-4}
		1F	2^{-5}
	F	1	2^{-1}
		3	2^{-2}
		7	2^{-3}
		F	2^{-4}
		1F	2^{-5}

Table 9. Round 1 difference probabilities

Word	Input difference ($\Delta_{x_6^2}$)	Output difference (δ)	$Pr[\delta\|\Delta]$ (approx)
x_3^3	1	1	2^{-1}
		3	2^{-2}
		7	2^{-3}
		F	2^{-4}
		1F	2^{-5}
	3	1	$2^{-1.32}$
		3	2^{-2}
		5	$2^{-4.32}$
		7	2^{-3}
		D	$2^{-5.32}$
		F	2^{-4}
		1D	$2^{-6.32}$
		1F	2^{-5}
	5	3	$2^{-2.32}$
		5	2^{-2}
		7	2^{-3}
		D	2^{-3}
		F	2^{-4}
		1B	$2^{-6.32}$
		1D	2^{-4}
		1F	2^{-5}

Table 10. Round 3 difference probabilities

k_8, subtracts it from the ciphertext pairs, reverses the fourth round permute and mix, and calculates the difference $x_3^3(55) \oplus x_3'^3(55)$. If the difference equals the expected difference at a rate close to expected, the guess is a candidate. If not, then it cannot be correct and is eliminated as a candidate.

The target difference will change as the window size changes. In particular, the target difference is zero when $w < 10$. As the window size increases, the target difference will change as more bits are included in the window.

The adversary begins by taking a greedy approach and choose the highest probability output at each step, given the appropriate input differential. Tables 8 to 10 show that 1 is the highest probability outcome at each step: that is, the most likely difference at each step is a difference only in the last bit of the string being considered. However, all paths that result in the target difference contribute to the actual probability. That is, one must calculate the total probability (equation 10) for each round to find the probability of output difference δ occurring given all possible input differences, Δ_i.

$$Pr[\delta] = \sum_i Pr[\delta|\Delta_i]Pr[\Delta_i] \tag{10}$$

The window size will start small, and gradually increase until all of both k_1 and k_8 have been guessed. Let us start with $w = 4$. Then 8 bits of key need to be guessed for the target window. Since the target window starts at position 55, and $55 + 4 - 1 < 64$, the target difference in the window is 0. Once the target window includes the least significant bit, the target difference will be 1.

Empirical results are shown in table 11. Once the adversary knows 9 bits of k_1 and 9 bits of k_8, increasing w to include new bits will not help the adversary gain new information. This happens because the newly guessed key bits are positioned such that they do not affect the differential, and therefore will not be distinguishable from incorrect key bits.

This could be remedied by using a longer difference pattern, such as `0x1F`. Although this difference pattern occurs with lower probability, it allows the adversary to find more of the key. In particular, the adversary

w	Δ_3	Probability with right key	Key bits right	Probability with wrong key
4	0	1	0	0.562455
8	0	1	4	0.781984
12	1	0.375138	4	0.256083
12	1	0.375138	8	0.307487
16	1	0.333555	8	0.237504
16	1	0.333555	9	0.238292
16	1	0.333555	10	0.333471

Table 11. Probabilities for $\Delta_{x_3^3} = 1$

can find up to 14 bits of each key word using this difference. Observed probabilities for this difference are given in table 12.

w	Δ_3	Probability with right key	Key bits right	Probability with wrong key
4	0	1	0	0.562455
8	0	1	4	0.781897
12	7	0.250082	8	0.244532
16	1F	0.030764	12	0.026866
20	1F	0.030572	16	0.030526

Table 12. Probabilities for $\Delta_{x_3^3} = 1F$

6.4 Diffusion

In this section, the application of metrics discussed in section 5.1 to Threefish-512 (version 2) [9] are compared. There are 512 bits, and thus 512 possible windows, in the state. For computing linear and pseudo-linear branch numbers, only the windows that start in the least significant bit of a word are considered here as active at the input to the second *Round*. Furthermore, only words with even index (0, 2, 4, 6) were used due to the higher window dependency of odd-indexed words.

The computation for differential branch numbers is done in a slightly different manner. Because branch number is concerned with the lower bound of diffusion, minimum diffusion should be ensured to derive the bound as accurately as possible. To this end, a one-bit difference shall be used in even index words.

The Threefish round function does not exactly fit the definition of a key-alternating round function presented in [7], but can easily be converted to one. First, the key injection is performed by addition modulo 2^n rather than exclusive-or. Since carry propagation is not considered to change activity, this does not impact the branch number or variants proposed in this paper. The main concern is that instead of applying one key-independent function and then one key injection in alternation, Threefish performs four key-independent functions and then one key injection.

Let *round* be the key-independent portion of a Threefish round function, and write $Round = round \circ round \circ round \circ round$. Now the alternation of *Round* and key injection follows the definition of a key-alternating block cipher.

Branch Number: Word View Consider a two-round trail that has a difference in least significant bit of the first word, depicted in figure 17. The input bundle weights (`state 0`) are $w_b(a \oplus b) = 1$, since only one word is active. At the input to the second *Round* (`state 1`), all eight words are active and therefore $w_b(Round(a) \oplus Round(b)) = 8$. It follows that $B_d(Round) \leq 9$. Note that the upper bound on the branch number in this case is 9.

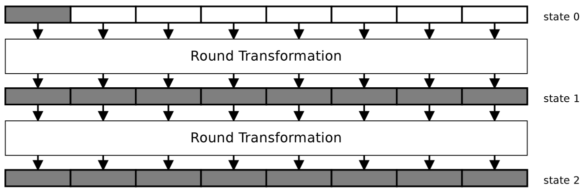


Fig. 17. Differential trail for Threefish-512, round 2 submission

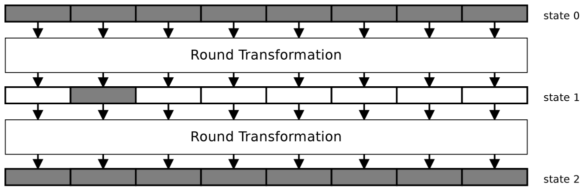


Fig. 18. Linear trail for Threefish-512, round 2 submission

Consider a linear approximation where one bit is active at the output of the first *Round* (`state 1`). Such an approximation is shown in figure 18. The 64-bit word encompassing that bit depends on all words at the input to the first *Round* (`state 0`). It follows that $B_l(Round) \leq 9$.

Branch Number: S-box Representation Using the S-box representation of modular addition (section 5.1) for Threefish-512, a different set of linear and differential branch numbers can be derived.

Consider the differential trail in figure 17, where the difference is located in the least significant bit of the first word. The S-box representation, where a byte is a bundle, is shown in figure 19. In the input to the first round, one bundle is active, so $w_b(a \oplus b) = 1$. At the input to the second *Round* (`state 1`), 21 bundles are active. Thus the bound $B_d(Round) \leq 22$.

In this view, with a byte as bundle, the upper bound for differential branch number is 65. Compare this to the differential branch number obtained by looking at 64-bit words in section 6.4. In section 6.4, all of `state 1` was active, whereas in the S-box representation, the same difference pattern only activates much less of the state. Using the S-box representation, it is apparent that the lower bound on diffusion is not as good as it was before. That is, a one-bit input difference in the S-box representation produced differential branch number of at most 22, whereas the word view produced a differential branch number of at most 9, where 9 is the maximum possible for the word model. Calculating the differential branch number in the S-box representation reveals more information about the propagation of an input difference by showing portions of the state where the difference does not propagate. Therefore, the S-box representation provides a more informative upper bound over the same difference pattern.

Figure 20 depicts a linear trail, with Threefish-512 using the S-box representation. At the input to the first *Round* (`state 0`), 26 bundles are active, giving $w_b(\alpha) = 26$. After one *Round* (`state 1`), there is only one bundle active, so $w_b(\beta) = 1$. This gives an upper bound on the linear branch number of $B_l(Round) \leq 27$.

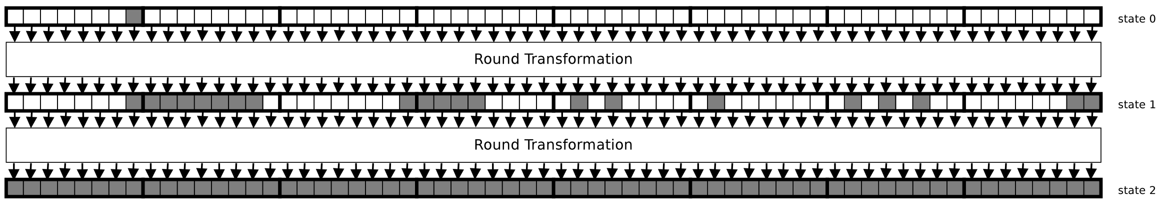


Fig. 19. Differential trail for Threefish-512, S-box representation

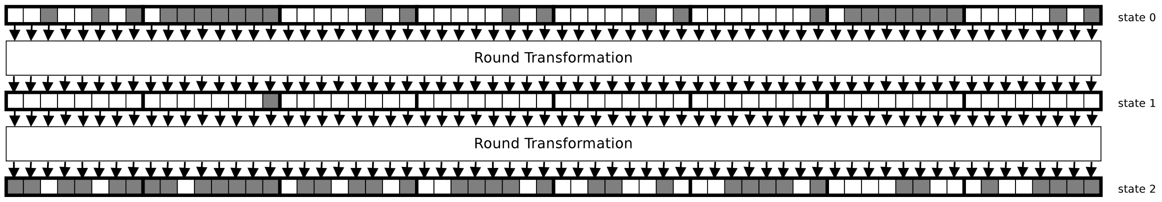


Fig. 20. Linear trail for Threefish-512, S-box representation

The pseudo-linear branch number can easily be derived from the linear approximation by increasing w. It was previously shown that this approximation can be distinguished from a random permutation when $w \geq 5$. This will not alter the number of active bundles in β, the input to the second *Round*, but it will change the number of active bundles of the input. Because a bundle is a byte of the state and the active 5-bit window of β starts at the least significant bit, only one bundle is active. On the other hand, the weight of the input selection pattern, α, increases because some 5-bit windows do not align with bundles. That is, one 5-bit window can span, and therefore activate, two bundles. This results in an input bundle weight $w_b(\alpha) = 32$, giving $B_{pl}(Round, 5) \leq 33$.

Effective Branch Number Let x be the input to the mix function and y be the output. Write x_l for the left input and x_r for the right input, and similar for y. $y_r = (x_l \boxplus x_r) \lll constant$, but $(x_l \boxplus x_r)$ can be written as y_l. Therefore, we only consider each mix function to contain one addition.

Consider the 8 round trail of section 6.2, where a single window is computed in the forward and backward direction. There are 11 additions (excluding key additions) in the trail over the first *Round* (rounds 1-4). In *Round* 2 (rounds 5-8), there are 7 subtractions in the trail. Since the number of subtractions of the second *Round* is less than the number of additions in the first *Round* and branch number gives a lower bound, let $w = 7$. Overall, this yields $\lfloor lg(7) \rfloor = 2$ and $w = 4$.

Consider an input difference of 1 in the least significant bit of the first word, with the same differential trail as in figures 17 and 19. Then $ew_b(a \oplus b) = \lceil \frac{1}{4} \rceil = 1$. At the input to the second *Round*, 111 active bits are in the trail, giving an effective bundle weight $ew_b(Round(a) \oplus Round(b)) = \lceil \frac{111}{4} \rceil = 28$. This gives a bound on effective differential branch number, $EB_d(Round) \leq 29$.

Consider the approximation of section 6.2. At the input to the first *Round*, 113 bits are part of the trail. This gives effective bundle weight $ew_b(\alpha) = 29$. There are four active bits (one window) at the output of *Round*, giving $ew_b(\beta) = 1$. Therefore, $EB_{pl}(Round) \leq 30$.

7 Conclusion

This paper has shown that pseudo-linear approximations may be applicable to round functions that are composed of addition modulo 2^n, exclusive-or, rotation, and word permutations. When permutation is performed on the word level, approximations may be constructed that apply to windows with size less than or equal to the word size. In particular, pseudo-linear approximations trace dependencies of w-bit target windows throughout a function, using addition modulo 2^w to approximate addition modulo 2^n. Reducing n-bit word addition to w-bit word addition allows the adversary to reduce the number of key bits that need to be guessed, and thus reduces overall attack complexity. The carry function will either add one to a window or not, depending on the sum of preceding bits. The carry function is independent of the window size. By increasing the size of the window, the expected number of times an approximation holds at random decreases, while the number of times the approximation holds for the cipher remains unchanged. Hence, the bias increases. It is important to note that, as with traditional linear cryptanalysis, high diffusion and a large enough number of rounds can be used to cause approximations over an entire cipher to become infeasible. In this situation, too many key bits would typically need to be guessed, and the search space would swiftly approach that of a brute force search.

This work has also shown that the approximations may be used to find truncated differentials. In addition, the branch number metric has been revisited to provide a more focused idea of diffusion in functions that use large words.

It would be interesting to understand the generality of such attacks, and also the types of ciphers that are more and less resilient to these attacks. The following are among open questions that come from this work. First, a question arises that is similar to that in linear cryptanalysis, where one computes the parity of some key bits from related input and output bits. Is there a simple function of the internal key bits in the pseudo-linear approximations of several ARX rounds? If such a function were found, the adversary would need only to compute this function, rather than find the key bits themselves. Second, how can one improve carry pattern selection for different input distributions? Third, are there other properties of linearity that can be leveraged to improve bias? Fourth, are larger words more or less secure than smaller words?

References

1. Aumasson, J.-P., Calik, C., Meier, W., Ozen, O., Phan, R. C.-W., and Varici, K. Improved cryptanalysis of Skein. Cryptology ePrint Archive, Report 2009/438, 2009. Available at `http://eprint.iacr.org/`.
2. Baignères, T., Stern, J., and Vaudenay, S. Linear cryptanalysis of non binary ciphers. In *Selected Areas in Cryptography* (2007), C. M. Adams, A. Miri, and M. J. Wiener, Eds., vol. 4876 of *LNCS*, Springer, pp. 184–211.
3. Chabaud, F., and Vaudenay, S. Links between differential and linear cryptoanalysis. In *EUROCRYPT* (1994), pp. 356–365.
4. Chen, J., and Jia, K. Improved related-key boomerang attacks on round-reduced Threefish-512. Cryptology ePrint Archive, Report 2009/526, 2009. Available at `http://eprint.iacr.org/`.
5. Cho, J. Y., and Pieprzyk, J. Multiple modular additions and crossword puzzle attack on NLSv2. In *ISC* (2007), J. A. Garay, A. K. Lenstra, M. Mambo, and R. Peralta, Eds., vol. 4779 of *LNCS*, Springer, pp. 230–248.
6. Daemen, J., and Rijmen, V. The wide trail design strategy. In *Cryptography and Coding, 8th IMA International Conference, Cirencester, UK, December 17-19, 2001, Proceedings* (2001), B. Honary, Ed., vol. 2260 of *Lecture Notes in Computer Science*, Springer, pp. 222–238.
7. Daemen, J., and Rijmen, V. *The Design of Rijndael.* Springer, 2002, ch. 9, pp. 123–133.
8. Ferguson, N., Lucks, S., Schneier, B., Whiting, D., Bellare, M., Kohno, T., Callas, J., and Walker, J. The Skein hash function family (version 1.0), October 2008. Available at `http://www.skein-hash.info/sites/default/files/skein.pdf`.
9. Ferguson, N., Lucks, S., Schneier, B., Whiting, D., Bellare, M., Kohno, T., Callas, J., and Walker, J. The Skein hash function family (version 1.1), November 2008. Available at `http://www.skein-hash.info/sites/default/files/skein1.1.pdf`.
10. Ferguson, N., Lucks, S., Schneier, B., Whiting, D., Bellare, M., Kohno, T., Callas, J., and Walker, J. The Skein hash function family (version 1.3), October 2010. Available at `http://www.skein-hash.info/sites/default/files/skein1.3.pdf`.
11. Jr., B. S. K., and Robshaw, M. J. B. Linear cryptanalysis using multiple approximations. In *CRYPTO* (1994), vol. 839 of *LNCS*, Springer, pp. 26–39.
12. Kelsey, J., Schneier, B., and Wagner, D. Mod n cryptanalysis, with applications against RC5p and m6. In *FSE* (1999), L. R. Knudsen, Ed., vol. 1636 of *LNCS*, Springer, pp. 139–155.
13. Khovratovich, D., and Nikolic, I. Rotational cryptanalysis of ARX. In *FSE 2010* (2010), LNCS, Springer.
14. Knudsen, L. R. Truncated and higher order differentials. In *FSE* (1994), vol. 1008 of *LNCS*, Springer, pp. 196–211.
15. Lai, X., and Blahut, R. E. Higher order derivatives and differential cryptanalysis. In *Proc. Symp. Communication, Coding and Cryptography* (1994), Kluwer, pp. 227 – 233.
16. Langford, S. K., and Hellman, M. E. Differential-linear cryptanalysis. In *CRYPTO* (1994), vol. 839 of *LNCS*, Springer, pp. 17–25.
17. Liskov, M., Rivest, R. L., and Wagner, D. Tweakable block ciphers. In *CRYPTO* (2002), M. Yung, Ed., vol. 2442 of *LNCS*, Springer, pp. 31–46.
18. Matsui, M. Linear cryptanalysis method for DES cipher. In *EUROCRYPT* (1993), T. Helleseth, Ed., vol. 765 of *LNCS*, Springer, pp. 386–397.
19. McKay, K. A., and Vora, P. L. Pseudo-linear approximations for ARX ciphers with application to Threefish. In *Second SHA-3 Candidate Conference* (August 2010).

20. McKay, K. A., and Vora, P. L. Pseudo-linear approximations for arx ciphers: With application to threefish. Cryptology ePrint Archive, Report 2010/282, 2010. `http://eprint.iacr.org/`.
21. Nyberg, K. Linear approximation of block ciphers. In *EUROCRYPT* (1994), pp. 439–444.
22. Nyberg, K. Correlation theorems in cryptanalysis. *Discrete Applied Mathematics 111*, 1-2 (2001), 177–188.
23. Nyberg, K., and Wallén, J. Improved linear distinguishers for snow 2.0. In *FSE* (2006), M. J. B. Robshaw, Ed., vol. 4047 of *LNCS*, Springer, pp. 144–162.
24. Staffelbach, O., and Meier, W. Cryptographic significance of the carry for ciphers based on integer addition. In *CRYPTO* (1990), A. Menezes and S. A. Vanstone, Eds., vol. 537 of *LNCS*, Springer, pp. 601–614.
25. Wagner, D. The boomerang attack. In *FSE* (1999), L. R. Knudsen, Ed., vol. 1636 of *Lecture Notes in Computer Science*, Springer, pp. 156–170.
26. Wallén, J. Linear approximations of addition modulo 2^n. In *Fast Software Encryption 2003* (2003), vol. 2887 of *LNCS*, Springer-Verlag, pp. 261–273.
27. Wallén, J. Linear approximations of addition modulo 2^{n}. In *FSE* (2003), T. Johansson, Ed., vol. 2887 of *LNCS*, Springer, pp. 261–273.
28. Wallén, J. On the differential and linear properties of addition. Research Report A84, Helsinki University of Technology, Laboratory for Theoretical Computer Science, Espoo, Finland, December 2003.
29. Wu, H., and Preneel, B. Cryptanalysis of the stream cipher ABC v2. In *Selected Areas in Cryptography* (2006), E. Biham and A. M. Youssef, Eds., vol. 4356 of *LNCS*, Springer, pp. 56–66.

A Addition Masks for Approximations

Round	**Mix 0**	**Mix 1**	**Mix 2**	**Mix 3**
1	0, 3, 14, 20, 31, 34, 47, 51	0, 3, 20, 22, 47	0, 20	0, 20
2	0, 3, 20, 47	0	0	0
3	0, 20	0	–	–
4	0	–	–	–
5	–	–	–	–
6	–	–	–	39
7	–	–	39, 56	52
8	21	14, 31, 39, 56	31, 52	13

Table 13. Addition masks for 8 round approximation, mix functions

Round	Mix 0	Mix 1	Mix 2	Mix 3
1	0, 1, 3, 7, 8, 10, 12, 14, 15, 18 ,20, 21, 23, 25, 26, 28, 31, 32, 34, 38, 39, 40, 43, 45, 46, 47, 51, 54, 55, 56, 59, 63	0, 3, 7, 8, 9, 10, 12, 15, 20, 22, 23, 25, 28, 30, 31, 32, 34, 37, 40, 45, 47, 51, 54, 55, 59	0, 1, 7, 8, 10, 12, 14, 16, 20, 24, 25, 26, 28, 31, 32, 37, 38, 40, 45, 50, 51, 55, 56, 60	0, 1, 4, 7, 8, 10, 12, 13, 16, 20, 24, 25, 28, 29, 31, 32, 37, 38, 40, 45, 49, 51, 53, 55, 56, 62
2	0, 3, 7, 8, 12, 15, 20, 25, 28, 31, 32, 34, 40, 45, 47, 51, 54, 55, 59	0, 1, 7, 12, 16, 20, 25, 31, 32, 40, 45, 51, 55, 56	0, 8, 10, 12, 25, 28, 38, 40, 51	0, 8, 10, 12, 15, 23, 25, 40, 51
3	0, 7, 12, 20, 25, 32, 45, 51	0, 8, 12, 25, 51	0, 10, 40, 51	0, 31, 40, 51, 55
4	0, 12, 25, 51	0, 40, 51	0	0
5	0, 51	0	0	0
6	0	0	–	–
7	0	–	–	–
8	–	–	–	0
9	–	–	0, 22	–
10	58	0, 19, 22, 41	–	–
11	0, 4, 19, 22, 27, 41, 46, 49	33, 55	36	27, 58
12	18, 21, 33, 40, 55	5, 8, 27, 36	2, 21, 24, 27, 33, 43, 58	0, 2, 4, 17, 19, 21, 22, 27, 36, 39, 41, 44, 46, 49, 58, 63

Table 14. Addition masks for 12 round approximation, mix functions