Pseudo-Linear Approximations for ARX Ciphers
With Application to Threefish

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

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

Abstract. The operations addition modulo 2^n and exclusive-or have recently been combined to obtain an efficient mechanism for nonlinearity in block cipher design. In this paper, we show that ciphers using this approach may be approximated by *pseudo-linear* expressions relating groups of contiguous bits of the round key, round input, and round output. The bias of an approximation can be large enough for known plaintext attacks. We demonstrate an application of this concept to a reduced-round version of the Threefish block cipher, a component of the Skein entry in the secure hash function competition.

Keywords: Linear cryptanalysis, non-linear approximations, Threefish, Skein

1 Introduction

Modern block ciphers are expected to be resilient to linear cryptanalysis[1], one of the most powerful known attacks against block ciphers. The attack relies on the existence of linear distinguishers of the cipher over $GF(2)$. A linear distinguisher is a linear equation over $GF(2)$ relating bits of the plaintext, ciphertext, and key that holds with probability distinct from $\frac{1}{2}$. In order to prevent linear cryptanalysis from being effective, some block ciphers have employed a combination addition modulo 2^n with exclusive-or. If one examines addition modulo 2^n in terms of its effect on bits, one observes that the carry function provides non-linearity; that is, the carry function is not expressed as an exclusive-or function. Our contribution is a method for approximating functions that use a combination of addition modulo 2^n, rotation, and exclusive-or (ARX). In this paper we demonstrate that the combination of addition modulo 2^n and exclusive-or is not immune to efficient approximation – even if the word size is large – and describe pseudo-linear approximations. We illustrate the use of these approximations on reduced-round variants of Threefish [2]. We present an 8 round approximation and use it in a key recovery attack on 11 rounds of Threefish-256, without whitening. We also present a 12 round approximation that could be used on 15 rounds of Threefish-256, without whitening.

We examine a window (grouping of contiguous bits) of size w for $w < n$, and make the following simple observations. First, addition modulo 2^n on the window can be approximated by addition modulo 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 depend exclusively on the probability distribution of the carry, and this probability is independent of w. Third, the probability of correctness for a random guess of the value of the window decreases exponentially with w. Hence, the influence of the carry decreases as w increases. Finally, the bias of our approximation (the difference between the probability of correctness of our approximation which depends completely on the carry and the probability of correctness of a random guess) increases with w. Using these basic facts, we show that simple non-linear approximations (that are *pseudo-linear* in that they are composed of exclusive-or and addition modulo 2^w operations for $w \geq 1$) are applicable to ARX ciphers. In particular, we show that the most likely values of the output of such ciphers lie in an interval. We also show that when data is permuted in large groups of contiguous bits, an approximation can address different window sizes with trivial modifications to the approximation.

This paper is organized as follows. Section 2 describes related work. Section 3 presents our method of approximation, and section 4 illustrates it using a reduced-round version of Threefish. Section 5 concludes.

* 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 Related Work

Several interesting properties of addition have been derived [3][4][5][6]. In particular, the linear properties of addition and subtraction modulo 2^n reduce to the linear properties of the carry function[5]. The addition of two numbers modulo 2^n can be expressed as $x \oplus y \oplus carry(x, y)$[6], where the carry function can be defined recursively[5], or as $(x + y) \oplus x \oplus y$[7]. Addition modulo 2^n can be approximated by the inner product of k-tuples of vectors representing inputs/outputs and linear masks [8].

The bias of linear relationships over integer addition are particularly important. The probability distribution of the carry in addition with an arbitrary number of inputs has been explored[3]. The carry function is biased, and has been used to form linear relationships between a bit in a sum and the carry-out from addition of previous bits[9]. Bit-linear relationships between contiguous bits using this bias have also been shown[10]. The probability distribution of the carry function in integer addition has been found to yield different probabilities for even and odd number of inputs [3].

Multiple linear approximations have been used to perform linear cryptanalysis [11]. It has also previously been proposed that non-linear approximations can replace linear approximations in linear cryptanalysis [12]. Linear cryptanalysis has been generalized to include non-binary ciphers [13]. In particular, it has been extended to apply to any additive group $\mathbb{Z}_m^r$. The method was applied to SAFER[14], which uses a mixture of addition operations in $\mathbb{Z}_2$ and $\mathbb{Z}_{256}$. This method was not applied to ARX ciphers with large word size, such as 64-bit words using addition in $\mathbb{Z}_{2^{64}}$.

Mod n cryptanalysis[15] is a partitioning attack that exploits weaknesses in ciphers based on addition and rotation. An application of the technique showed that RC5 depends on the mixture of addition and exclusive-or for security.

Rotational cryptanalysis [16] is a universal related-key technique that is applicable to add, rotate, xor (ARX) ciphers. It follows the propagation of rotational pairs $(X, X \gg r)$ throughout the cipher, where addition modulo 2^n destroys these pairs with some probability. It was shown to be effective for related-key attacks. At present, it is the strongest attack against Threefish.

The Skein hash function family[2] is currently under review as a candidate for the SHA-3 family of hash functions. Skein contains a tweakable block cipher[17] called Threefish. Four and eight round differential attacks on Threefish were presented in the submission, but linear attacks were not addressed beyond noting that nonlinearity is provided by mixing the use of addition and exclusive or.

3 Pseudo-Linear Cryptanalysis

Linear cryptanalysis traditionally applies to linear equations in $GF(2)$. Instead of single bits, we consider larger groupings of contiguous bits, which we call *windows*. Consider the set G of all possible windows of size w—that is, all 2^w possible bits-strings of length w. We require the following two operations on G: bitwise exclusive-or and addition modulo 2^w. The two operations do not distribute and hence G, with the two operations, is not a ring. We refer to this as a *pseudo-linear system*.

We may not be able to form linear equations over this system, but we can make approximations in these pseudo-linear systems using windows of length $w < n$. Instead of performing addition modulo 2^n, approximations can be created with addition modulo 2^w. This is useful for two reasons: (1) a carry into an addition window from the addition of less significant bits preceding the window has the same effect regardless of the window size, and (2) the success probability of a random guess decreases as window size increases. This also means that the bias of our approximations increases with w.

To illustrate why this is an improvement over traditional linear cryptanalysis, consider the following example depicted in figure 1. There are two n-bit words, added modulo 2^n. Suppose only the value of the dark square, labeled c, is needed. a and b are the operand windows in the same position. The square contains w bits, and c can be approximated as $a \boxplus_w b$. Whether the approximation is correct or not will depend on the value of the carry into the square. The probability that the carry is zero—and the approximation is correct—is approximately $\frac{1}{2}$, and approximately equal to the probability that the carry is one and the approximation is incorrect. If $w = 1$, the probability of guessing c correctly at random is also $\frac{1}{2}$ and there

is no benefit to approximating c by $a \boxplus_w b$. On the other hand, if w is large, the probability of guessing the value of c correctly at random is $\frac{1}{2^w}$, and the approximation, with a correctness probability of $\frac{1}{2}$, provides considerable advantage. *Thus, by looking at windows of several bits rather than a single bit, the effect of the carry is diminished, and one is able to approximate several bits with high bias.*

Any window in an ARX function can be approximated using pseudo-linear equations. Such approximations are called pseudo-linear approximations.

3.1 Notation

The following notation is used in this paper.

$\oplus$ Exclusive-or
$\boxplus$ Addition modulo 2^n
$\boxplus_i$ Addition modulo 2^i
$\lll$ ($\ggg$) Left (right) rotation on an n-bit word
$\ll$ ($\gg$) Left (right) shift on an n-bit word

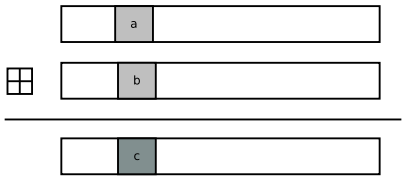

Fig. 1. Window addition

3.2 Addition Windows: Simple Analytical Results

Several properties are useful for predicting carries. We start by assuming that the n-bit words that are added come from a uniformly random distribution. We also assume throughout this paper that words are represented most significant bit first.

Lemma 1. *Consider two* n*-bit words,* a *and* b*, selected uniformly at random, and a window size* w*. Let* $part(\mathrm{a}, \mathrm{s}, \mathrm{e})$ *be a function which returns the bits of* a *in the range* [s, e)*, where the range* [0,w) *represents the least significant* w *bits and* $\mathrm{s} < \mathrm{e} \leq \mathrm{n}$*. Then* $part(\mathrm{a} \boxplus \mathrm{b}, \mathrm{i}, \mathrm{i}+\mathrm{w}) = part(\mathrm{a}, \mathrm{i}, \mathrm{i}+\mathrm{w}) \boxplus_w part(\mathrm{b}, \mathrm{i}, \mathrm{i}+\mathrm{w})$ *with probability greater than* $\frac{1}{2}$*.*

Proof. If there is no carry into bit i, then the equation is true. If there is a carry into bit i, then the equation is false. Let $j = i - 1$. For the addition of two j-bit integers, there are $2^{2j-1} + 2^{j-1}$ of 2^{2j} operand combinations that will not produce a carry in the output. Therefore the probability that the carry into bit is 0 i is $\frac{1}{2} + 2^{-j-1}$, which agrees with the bias presented in [9].

Corollary 1 $part(\mathrm{a} \boxplus \mathrm{b}, 0, \mathrm{w}) = part(\mathrm{a}, 0, \mathrm{w}) \boxplus_w part(\mathrm{b}, 0, \mathrm{w})$ *with probability 1.*

Proof. Because this is the least significant window, there are no carry bits into the sum. Therefore the probability of the equation being true is 1.

Corollary 2 $part(\mathrm{a} \boxplus \mathrm{b}, \mathrm{i}, (\mathrm{i}+\mathrm{w}) \bmod \mathrm{n}) = part(\mathrm{a}, \mathrm{i}, (\mathrm{i}+\mathrm{w}) \bmod \mathrm{n}) \boxplus_w part(\mathrm{b}, \mathrm{i}, (\mathrm{i}+\mathrm{w}) \bmod \mathrm{n})$ *with probability greater than* $\frac{1}{2}$*.*

Proof. The key 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. To accomplish this, we split the window in two: $part(a, i, n) \boxplus_w part(b, i, n)$ and $part(a, 0, (i + w) \bmod n) \boxplus_w part(b, 0, (i + w) \bmod n)$. The first sum is true by lemma 1 and the second is true by corollary 1.

Corollary 3 $part(\mathrm{a} \boxminus \mathrm{b}, \mathrm{i}, (\mathrm{i}+\mathrm{w}) \bmod \mathrm{n}) = part(\mathrm{a}, \mathrm{i}, (\mathrm{i}+\mathrm{w}) \bmod \mathrm{n}) \boxminus_w part(\mathrm{b}, \mathrm{i}, (\mathrm{i}+\mathrm{w}) \bmod \mathrm{n})$ *with probability greater than* $\frac{1}{2}$*.*

Proof. Let $a \boxminus b = c$. Then $b \boxplus c = a$, and $part(b \boxplus c, i, (i+w) \bmod n) = part(b, i, (i+w) \bmod n) \boxplus_w part(c, i, (i+w) \bmod n)$ with probability greater than $\frac{1}{2}$ by lemma 1 and corollary 2. Then we have $part(a, i, (i + w) \bmod n) = part(b, i, (i + w) \bmod n) \boxplus part(c, i, (i + w) \bmod n)$, which is the same as $part(a, i, (i + w) \bmod n) \boxminus part(b, i, (i + w) \bmod n) = part(c, i, (i + w) \bmod n)$, with probability greater than $\frac{1}{2}$.

Lemma 2. *part*(a $\oplus$ b, i, (i + w) mod n) = *part*(a, i, (i + w) mod n) $\oplus$ *part*(b, i, (i + w) mod n) *with probability 1.*

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

Rotations and shifts occur within a single word, and are easy to express as windows by shifting the indices. For example, $part(a \ggg c, i, (i+w) \bmod n) = part(a, (c+i) \bmod n, (c+i+w) \bmod n)$ and $part(a \gg c, i, i+w) = part(a, c+i, (c+i+w) \bmod n)$.

The addition window properties described above work quite well for window approximations where the operands are from a *uniformly random* distribution. Once an addition on this data is performed, the distribution is no longer random. It is easy to see why this is so. Consider four b-bit words sampled from a uniform distribution: a, b, a', b'. Let $c = a + b \bmod 2^k$, similarly c'. If there is a carry out of $a + b$, then c is biased towards the smaller values from 0 to $2^k - 1$. Similarly, if there is no carry out of $a + b$, c is biased towards the larger values from 0 to $2^k - 1$. Thus the carry out of $a + b$ and $a' + b'$ characterizes the distribution of c and c', and hence of the carry out of their sum.

As an example, consider two 4-bit words, a and b, and sum $c = a \boxplus b$. Let $a(i)$ be the 2-bit window with i denoting the position of the least significant bit. Suppose $w = 2$ and the goal is to approximate $c(2)$ (the approximation window has the third bit as its least significant bit). The carry into this addition window is determined entirely by the values in $a(0)$ and $b(0)$. There are 16 operand permutations, 6 of which produce a carry and 10 which do not. If there was not a carry into $c(2)$, then $Pr[c(0) = 0] = \frac{1}{10}$, $Pr[c(0) = 1] = \frac{2}{10}$, $Pr[c(0) = 2] = \frac{3}{10}$, and $Pr[c(0) = 3] = \frac{4}{10}$. If there was a carry into $c(2)$, then $Pr[c(0) = 0] = \frac{3}{6}$, $Pr[c(0) = 1] = \frac{2}{6}$, $Pr[c(0) = 2] = \frac{1}{6}$, and $Pr[c(0) = 3] = 0$.

Now consider $e = c \boxplus d$, where d is the sum of two other 4-bit values chosen at random. If $c(2) = a(2) \boxplus_2 b(2)$ (there was no carry), then there is a greater chance that $e(2) = c(2) \boxplus_2 d(2) \boxplus_2 1$. If $c(2) = a(2) \boxplus_2 b(2) \boxplus_2 1$, then there is a greater chance that $e(2) = c(2) \boxplus_2 d(2)$.

3.3 Approximations of ARX round functions

Base Approximations Using the windowing method described above, it is straightforward to create an approximation for an ARX round function by following the windows. An approximation that simply follows the windows is equivalent to assuming that the carry into all windows is 0. We refer to this approximation as the *base approximation*.

Although the carry is biased, one still expects a carry for approximately half of the window additions. That is, every other round of approximated window additions, one expects a carry into the window. To allow for a carry approximately every other round of approximation, we use *carry patterns* and *offsets*.

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, $base + C^j$, represents a distinct approximation. By applying several carry patterns to the same input/output pairs, one can increase the probability of a correct approximation, which in turn increases the bias.

Offsets Another method of compensating for mis-predicted carries is through the use of *offsets*. Consider an integer *offset*$\in \{0, \cdots, 2^{w-1} - 1\}$ that 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 2 [1]. The horizontal line shows the probability of guessing correctly at random (in this case, $\frac{1}{256}$).

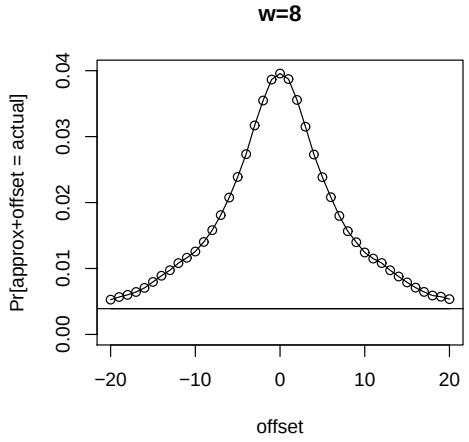

Fig. 2. Observed probability of correctly guessing window of ARX target over several offsets

Instead of considering an approximation correct only if it matches the target window, we consider it correct if it falls within the range $approx \pm offset$. In figure 2, the approximation is correct with $offset = 1$ with probability $0.039571 + 0.038652 + 0.038726 = 0.116949$. This will increase the number of correct guesses even though carries were guessed incorrectly.

Using this technique improves our results with lower overhead than the application of more carry patterns. However, carry patterns are still important. In particular, we observe that the carry pattern used determines the height of the curve, so a carefully chosen pattern improves the bias for approximations when the target falls in the range $approx \pm offset$. This is because guessing the carry pattern correctly is not equivalent to guessing a correct number to add at the end after the base case. (This number could be, for example, the sum of all the carries that were ignored.) Rotation changes the impact of a single carry, and the value of the carry affects the value of several later additions. Thus guessing the carry correctly each time is superior to guessing the effective sum of all the carries.

Empirical examples with Threefish-256 are presented in section 4.

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. Thus to get the bias, compute $bias = times\ correct - \frac{\#\ patterns}{2^w} \times pairs$. Similarly, when offsets are used, the bias is computed as $times\ correct - \frac{((2 \times offset)+1)}{2^w} \times pairs$. If both are used, $bias = times\ correct - \frac{\#\ patterns \times ((2 \times offset)+1)}{2^w} \times pairs$.

Flexibility The techniques described above lead to flexible approximations. Once a base approximation has been established, it can easily be modified through the use of offsets and carry patterns. It is also trivial to change the window size, since it is the start of the window that is important. The end of the window can be computed dynamically based on the desired w.

[1] The results in figure 2 were obtained from approximating $x(0)$ in the first word of the 8^{th} round of Threefish-256 with the base approximation. Further details in section 4.

Comparison to Linear Cryptanalysis Though this attack is clearly inspired by linear cryptanalysis, there is a difference that should be noted. Currently, we cannot concatenate and simplify the effect of several approximations. This is because the two operations — exclusive-or and addition modulo 2^w — do not commute. Because of this, we cannot combine all key bits 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 to be combined into a single function of key bits. Our approximation requires us to try all key bits used in the approximation. Even so, we are able to obtain attacks more efficient than brute force because our approximations enable us to reduce the number of key bits from those required by the cipher. We can approximate one operation using the other, but this only works reasonably well when w is small, which is precisely when pseudo-linear approximations provide the weakest assistance.

4 Application to Threefish-256

Threefish is a tweakable block cipher that is part of the Skein hash function family[2]. There are three variants designed for 256, 512, and 1024-bit blocks and keys. Instead of using substitution boxes, Threefish relies on combining addition modulo 2^{64} and exclusive-or to obtain non-linearity. In this paper, we examine [2] Threefish-256.

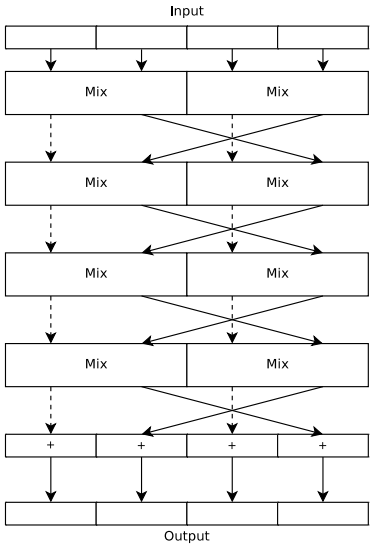

Fig. 3. Four rounds of Threefish-256

Threefish-256 is comprised of 72 rounds over a 256-bit block with a 256-bit key. State is maintained throughout the cipher in four 64-bit words.

Each round of Threefish-256 contains two parallel mix functions followed by a permutation. Each mix function takes two words as inputs, a and b, and outputs the words $a \boxplus b$ and $(a \boxplus b) \oplus (b \lll constant)$, where the rotation constant is specific to a particular mix function. The permutation (1 3) is then applied. That is, if we label the first state word as word 0 and the last state word as word 3, words 1 and 3 are swapped in the permutation function.

If we label the four input words as $x0$, $x1$, $x2$, $x3$, and four output words as $y0$, $y1$, $y2$, $y3$, respectively, a round of mix and permutation functions can be expressed as

[2] as described in the original submission to NIST

$$y0 = x0 \boxplus x1$$
$$y2 = x2 \boxplus x3$$
$$y1 = (x2 \boxplus x3) \oplus (x3 \lll r_{1,round-1})$$
$$y3 = (x0 \boxplus x1) \oplus (x1 \lll r_{0,round-1})$$

Every fourth round (4, 8, 12, etc.), a word of the round key is injected into each state word by addition modulo 2^{64}. Figure 3 depicts four rounds of Threefish-256.

Skein's UBI mode changes the key for each block, so a computationally intensive key schedule would have caused high performance overhead for the hash function. This was not desirable, so Threefish has a very simple key schedule and relies on high diffusion over many rounds to provide protection from attacks. The key schedule is generated from a combination of key words and tweak words. There are four 64-bit key words (k_0 to k_3) and a fifth parity word (k_4) derived by equation 1. There are two 64-bit tweak words (t_0 and t_1), and a tweak parity word ($t_2 = t_0 \oplus t_1$). In each key injection, four of the key words are used in conjunction with two of the tweak words. In this work, we fix the tweak schedule to all zeros. The resulting key schedule is shown in equation 2.

$$k_4 = \lfloor \frac{2^{64}}{3} \rfloor \oplus \bigoplus_{0 \leq i \leq 3} k_i \tag{1}$$

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

Because of the parity relation in the key schedule, it is easy to calculate a portion of one key word given the corresponding bits in the other four key words.

Using techniques described in the previous section, we can create approximations of windows throughout the cipher. Because the operations are performed on 64-bit words, $1 \leq w \leq 64$.

4.1 Threefish-256 Approximations

We present two approximations on modified versions of Threefish-256. The first is for the least significant window of the first state word after round 8. The second approximates several words in the output of round 12. Both modifications do not include the injection of a whitening key.

Our 12-round approximation can be used on up to 15 rounds of the modified Threefish-256. Because there is no whitening key, rounds 1 to 4, up to the key addition, are known. Similarly, decryption from rounds 15 through 13 is known because the key plays no role in those rounds. Effectively, this approximation covers the cipher from the key injection of round 4 to the key injection at round 12, inclusive.

The modified cipher for the 8-round approximation is similar. The cipher is reduced to 11 rounds, and decryption from rounds 11 to 8 is known because there are no key injections. Therefore 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. In other words, it represents the window returned by $part(x_i^j, m, (m + w) \bmod 64)$ since our windows grow from the least significant to most significant bit. In rounds that contain key injections, $x_i^j(m)^*$ represents the window after the permutation, but before the key injection occurs. $k_i(m)$ denotes the window starting at m of the i^{th} word of the key schedule.

The approximation for $x_0^8(0)$ is presented in table 1 as an example. It is broken down by round for clarity. 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 depend 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.

The 12-round approximation computes a window from both the plaintext and the ciphertext. In the encryption direction, we approximate the least significant window of the first word after the 10^{th} round,

$x_0^{10}(0)$. From the ciphertext, we then approximate the same word, denoted $d_0^{10}(0)$, and then check that $x_0^{10}(0) = d_0^{10}(0)$.

For the complete 12-round approximation, see appendix B.

$x_0^5(0) = (x_0^4(0)^* \boxplus_w k_1(0)) \boxplus_w (x_1^4(0)^* \boxplus_w k_2(0))$	$x_0^6(0) = x_0^5(0) \boxplus_w x_1^5(0)$
$x_0^5(22) = (x_0^4(22)^* \boxplus_w k_1(22)) \boxplus_w (x_1^4(22)^* \boxplus_w k_2(22))$	$x_0^6(22) = x_0^5(22) \boxplus_w x_1^5(22)$
$x_0^5(29) = (x_0^4(29)^* \boxplus_w k_1(29)) \boxplus_w (x_1^4(29)^* \boxplus_w k_2(29))$	$x_2^6(0) = x_2^5(0) \boxplus_w x_3^5(0)$
$x_2^5(0) = (x_2^4(0)^* \boxplus_w k_3(0)) \boxplus_w (x_3^4(0)^* \boxplus_w k_4(0))$	$x_3^6(0) = x_0^6(0) \oplus x_1^5(11)$
$x_2^5(11) = (x_2^4(11)^* \boxplus_w k_3(11)) \boxplus_w (x_3^4(11)^* \boxplus_w k_4(11))$	$x_3^6(22) = x_0^6(22) \oplus x_1^5(33)$
$x_2^5(22) = (x_2^4(22)^* \boxplus_w k_3(22)) \boxplus_w (x_3^4(22)^* \boxplus_w k_4(22))$	$x_1^6(0) = x_2^6(0) \oplus x_3^5(29)$
$x_2^5(33) = (x_2^4(33)^* \boxplus_w k_3(33)) \boxplus_w (x_3^4(33)^* \boxplus_w k_4(33))$	
$x_3^5(0) = x_0^5(0) \oplus (x_1^4(38)^* \boxplus_w k_2(38))$	$x_0^7(0) = x_0^6(0) \boxplus_w x_1^6(0)$
$x_3^5(29) = x_0^5(29) \oplus (x_1^4(3)^* \boxplus_w k_2(3))$	$x_2^7(0) = x_2^6(0) \boxplus_w x_3^6(0)$
$x_1^5(0) = x_2^5(0) \oplus (x_3^4(44)^* \boxplus_w k_4(44))$	$x_1^7(0) = x_2^7(0) \oplus x_3^6(22)$
$x_1^5(11) = x_2^5(11) \oplus (x_3^4(55)^* \boxplus_w k_4(55))$	
$x_1^5(22) = x_2^5(22) \oplus (x_3^4(2)^* \boxplus_w k_4(2))$	$x_0^8(0) = (x_0^7(0) \boxplus_w x_1^7(0)) \boxplus_w k_2(0)$
$x_1^5(33) = x_2^5(33) \oplus (x_3^4(13)^* \boxplus_w k_4(13))$	

Table 1. Approximation for $x_0^8(0)$, without whitening

4.2 Empirical Results for Approximations

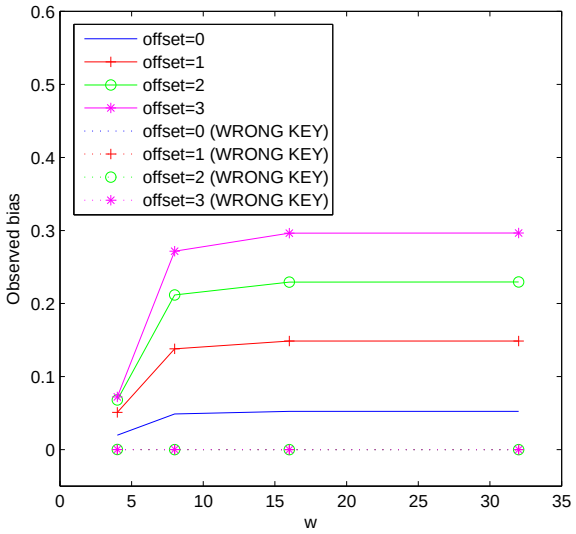

Fig. 4. Base approximation with window isolation, $x_0^8(0)$

We obtained empirical results for bias with varying window sizes, using 20,000,000 pseudorandomly generated (data, key) pairs and tweak schedule fixed to 0. We used [2], where differential biases were computed using 20,000,000 pseudorandom (data, key, tweak) tuples, as a model. Two implementation strategies were

used – window isolation and word isolation. Window isolation stores each window in a separate word. Word isolation stores all windows located the same word of the state in one word. For further details regarding implementation, see appendix A.

For simplicity, we represented carry pattern C^i as an array C^i_{jk}, where j is the round number and k is the word number. Each window in word k of round j was treated the same in these experiments.

Figure 4 demonstrates the effect of offsets on an approximation, with both the correct and incorrect key values. Offsets increase the probability of correctness for wrong values quite a bit when the range covers a large portion of the possible values, but the correct values still have a higher bias on average. The same is true when multiple carry patterns are used. Figure 4 also shows that there is a maximum bias that can be achieved with window isolation, where the maximum is less than 1.

Figure 5 demonstrates the improvements that can be gained by using word isolation over window isolation, when compared against figure 4 when *offset*=0. This figure also demonstrates that the bias grows to 1 as w increases using word isolation. Finally, it shows how different carry patterns can affect the approximation. The two carry patterns used here were:

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

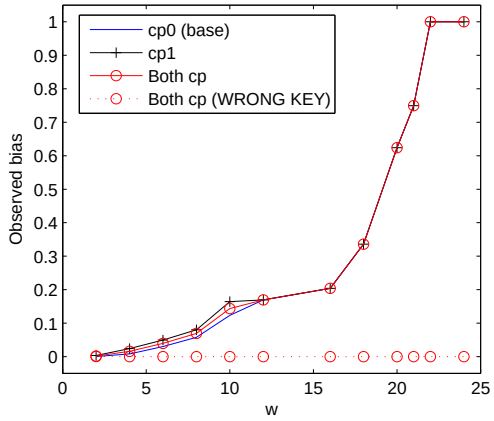

Fig. 5. $x^8_0(0)$ approximation with word isolation and different carry patterns (cp)

When enough key bits are guessed, the bias for different carry patterns converge (here, this happens at $w = 12$). If multiple carry patterns are used that have dissimilar biases, like the two used here, the bias of multiple patterns may be smaller than the bias obtained by using a single, stronger carry pattern.

Figure 6 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 bits of key need to be guessed. This approximation is computationally impractical for key recovery; however it may be possible to derive a better approximation that uses more of the same windows in rounds 4 and 12, thus reducing the number of key bits needed.

The empirical results presented in this section show that this method can be used to approximate a window with accuracy better than random. In the next section, we present a key recovery attack using the 8-round approximation.

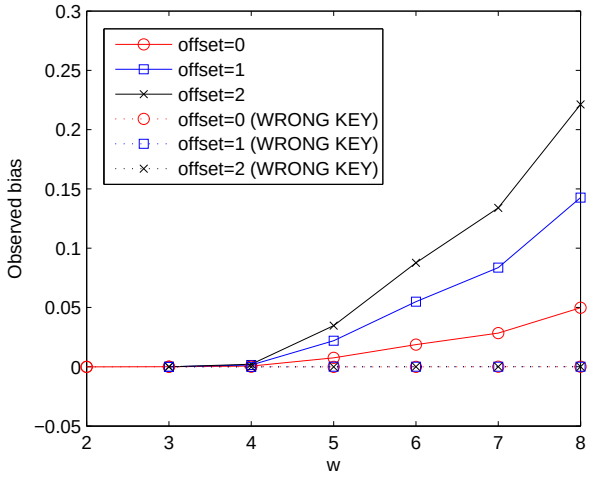

Fig. 6. 12 round approximation

4.3 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.

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 could be used for 11 rounds of Threefish-256, since there is not another key injection until round 12.

If $w = 3$, then the attack requires 58 bits of key (as compared to 256 bits for a brute force attack) to be guessed. Suppose that 40 bits needed in this attack were obtained by some other means, such as side channels. We show that the remaining 18 bits can be found using our approximations. In particular, we find the values of $k_1(0)$, $k_2(0)$, $k_3(0)$, $k_4(0)$, $k_2(3)$, and $k_4(11)$. This assumption that other bits are known prior to this attack was made due to limited time and resources. It is not a limitation of this technique.

Because the carry function is difficult to determine with incomplete knowledge of the operands, the correct value of the key does not always bubble to the top of the list. However, it should still be near the top. In these experiments, the goal was to eliminate 90% of the possible key values, with the hope that the correct value is in the remaining 10%. We also note when the correct key value has the maximum bias. 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 2.

	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 2. 500 attacks, 10,000 pairs

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 windows does not always help to effectively distinguish among correct and incorrect key bits. This can be seen in table 2 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.

5 Conclusion and Future Work

We have shown that pseudo-linear approximations are applicable to ARX ciphers. 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. By increasing the size of the window, the expected number of times a random guess is correct decreases, while the number of times the approximation for a window is correct remains unchanged. Hence, the bias increases. Eight and twelve round approximations for the Threefish-256 block cipher were presented as examples.

Our results show that pseudo-linear approximations can be used to distinguish an ARX function from a random permutation. These approximations can also be used for key recovery, as demonstrated in section 4.3.

It is important to note that, as with traditional linear cryptanalysis, high diffusion and a large enough number of rounds 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.

In ongoing work, we are applying the attack to Threefish-512 and Threefish-1024, and expect it to work until full diffusion is achieved in each variant. We plan to search for better approximations and improvements, and use them to mount key recovery attacks. This technique will also be applied to several hash functions and their building blocks, including Skein[2], Blue Midnight Wish[18], CubeHash [19], and BLAKE[20].

It would be interesting to understand the generality of such attacks, and the types of ciphers that are more and less resilient to these attacks. It would also be interesting to understand if linearity over a larger field is of use in the analysis of other primitives.

References

1. Matsui, M.: Linear cryptanalysis method for des cipher. In: Advances in Cryptology EUROCRYPT 93. Volume 765 of LNCS., Springer-Verlag (1993) 386–397
2. Ferguson, N., Lucks, S., Schneier, B., Whiting, D., Bellare, M., Kohno, T., Callas, J., Walker, J.: The skein hash function family (version 1.0) (October 2008) http://www.skein-hash.info/sites/default/files/skein.pdf.
3. Staffelbach, O., Meier, W.: Cryptographic significance of the carry for ciphers based on integer addition. In Menezes, A., Vanstone, S.A., eds.: CRYPTO. Volume 537 of LNCS., Springer (1990) 601–614
4. Wallén, J.: Linear approximations of addition modulo 2^n. In Johansson, T., ed.: FSE. Volume 2887 of LNCS., Springer (2003) 261–273
5. 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)
6. Lipmaa, H., Moriai, S.: Efficient algorithms for computing differential properties of addition. In Matsui, M., ed.: FSE. Volume 2355 of LNCS., Springer (2001) 336–350
7. Brier, E., Khazaei, S., Meier, W., Peyrin, T.: Linearization framework for collision attacks: Application to cubehash and md6. Cryptology ePrint Archive, Report 2009/382 (2009) urlhttp://eprint.iacr.org/.
8. Nyberg, K., Wallén, J.: Improved linear distinguishers for snow 2.0. In Robshaw, M.J.B., ed.: FSE. Volume 4047 of LNCS., Springer (2006) 144–162
9. Wu, H., Preneel, B.: Cryptanalysis of the stream cipher abc v2. In Biham, E., Youssef, A.M., eds.: Selected Areas in Cryptography. Volume 4356 of LNCS., Springer (2006) 56–66
10. Zhang, H., Wang, S., Wang, X.: The probability advantages of two linear expressions in symmetric ciphers (2006) Available at http://www.ecrypt.eu.org/stream/papersdir/2006/046.pdf.
11. Jr., B.S.K., Robshaw, M.J.B.: Linear cryptanalysis using multiple approximations. In Desmedt, Y., ed.: CRYPTO. Volume 839 of LNCS., Springer (1994) 26–39

12. Knudsen, L.R., Robshaw, M.J.B.: Non-linear approximations in linear cryptoanalysis. In Maurer, U.M., ed.: EUROCRYPT. Volume 1070 of LNCS., Springer (1996) 224–236
13. Baignères, T., Stern, J., Vaudenay, S.: Linear cryptanalysis of non binary ciphers. In Adams, C.M., Miri, A., Wiener, M.J., eds.: Selected Areas in Cryptography. Volume 4876 of LNCS., Springer (2007) 184–211
14. Massey, J.L.: Safer k-64: A byte-oriented block-ciphering algorithm. In Anderson, R.J., ed.: FSE. Volume 809 of LNCS., Springer (1993) 1–17
15. Kelsey, J., Schneier, B., Wagner, D.: Mod n cryptanalysis, with applications against rc5p and m6. In Knudsen, L.R., ed.: FSE. Volume 1636 of LNCS., Springer (1999) 139–155
16. Khovratovich, D., Nikolic, I.: Rotational cryptanalysis of arx. Preproceedings of FSE 2010 (2010)
17. Liskov, M., Rivest, R.L., Wagner, D.: Tweakable block ciphers. In Yung, M., ed.: CRYPTO. Volume 2442 of LNCS., Springer (2002) 31–46
18. Gligoroski, D., Klima, V., Knapskog, S.J., El-Hadedy, M., Amundsen, J., Mjolsnes, S.F.: Cryptographic hash function blue midnight wish. Submission to NIST (Round 2) (2009)
19. Bernstein, D.J.: Cubehash specification (2.b.1). Submission to NIST (Round 2) (2009)
20. Aumasson, J.P., Henzen, L., Meier, W., Phan, R.C.W.: Sha-3 proposal blake. Submission to NIST (2008)

A Window Implementation

We have implemented our windows in two different ways:

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

We find that the first is useful for developing approximations, but the latter is much better in practice. This is because the first case provides window isolation that allows easier debugging. However, when windows are adjacent or overlap, there is carry information that is not used due to the high level of isolation. In the second method, we take advantage of that extra 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 added. 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 63 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 the their correct value in the corresponding position, and all other bits set to 0. For example, $x(4)$ =0x2E would be represented as 00000000 00002E00. With word isolation and two 16-bit windows, $x(8) = 0$x5678 and $x(40) = 0$x1234, 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. Both techniques were used in these experiments. In particular, inactive bits were zeroed out for the key recovery attack, and were not altered in figure 4. We found the first approach to be more beneficial overall, although there were drops in bias at certain w values.

B 12-round Approximations

The full 12-round approximation described is detailed below. It is broken down by round for clarity. It is shown in word isolation format, where $x_i^j(a, b, c)$ indicates windows a, b, and c in word i of round j. To simplify the expressions, win_i^j is used to represent long lists of windows in x_i^j.

$win_0^5 = \{0, 3, 5, 8, 13, 14, 18, 22, 27, 29, 30, 32, 35, 37, 43, 53\}$
$x_0^5(win_0^5) = x_0^{4*}(win_0^5) \boxplus_w x_1^{4*}(win_0^5) \boxplus_w k_1(win_0^5) \boxplus_w k_2(win_0^5)$
$win_2^5 = \{0, 3, 5, 8, 11, 13, 14, 16, 19, 22, 24, 27, 30, 33, 36, 38, 41, 46, 53\}$
$x_2^5(win_2^5) = x_2^{4*}(win_2^5) \boxplus_w x_3^{4*}(win_2^5) \boxplus_w k_3(win_2^5) \boxplus_w k_4(win_2^5) \boxplus_w 1$
$win_3^4 = \{44, 49, 52, 55, 57, 58, 60, 63, 2, 4, 7, 10, 13, 15, 18, 21, 26\}$
$win_1^5 = \{0, 5, 8, 11, 13, 14, 16, 19, 22, 24, 27, 30, 33, 35, 38, 41, 46\}$
$x_1^5(win_1^5) = x_2^5(win_1^5) \oplus ROTL(x_3^4(win_3^{4*}) \boxplus_w k_4(win_3^4), 20)$
$win_1^4 = \{38, 41, 43, 46, 51, 52, 56, 3, 6, 11, 17, 27\}$
$win_3^5 = \{0, 3, 5, 8, 13, 14, 18, 29, 32, 37, 43, 53\}$
$x_3^5(win_3^5) = x_0^5(win_3^5) \oplus ROTL(x_1^{4*}(win_1^4) \boxplus_w k_2(win_1^4), 26)$

$win_0^6 = \{0, 5, 8, 13, 14, 22, 27, 30, 35\}$
$x_0^6(win_0^6) = x_0^5(win_0^6) \boxplus_w x_1^5(win_0^6)$
$win_2^6 = \{0, 3, 5, 8, 13, 14, 53\}$
$x_2^6(win_2^6) = x_2^5(win_2^6) \boxplus_w x_3^5(win_2^6)$
$win_1^6 = \{0, 3, 8, 14, 53\}$
$x_1^6(win_1^6) = x_2^6(win_1^6) \oplus ROTL(x_3^5, 35)$
$win_3^6 = \{0, 5, 8, 13, 22, 27, 30, 35\}$
$x_3^6(win_3^6) = x_0^6(win_3^6) \oplus ROTL(x_1^5, 53)$

$win_0^7 = \{0, 8, 14\}$
$x_0^7(win_0^7) = x_0^6(win_0^7) \boxplus_w x_1^6(win_0^7)$
$win_2^7 = \{0, 5, 8, 13\}$
$x_2^7(win_2^7) = x_2^6(win_2^7) \boxplus_w x_3^6(win_2^7)$
$win_1^7 = \{0, 5, 8, 13\}$
$x_1^7(win_1^7) = x_2^7(win_1^7) \oplus ROTL(x_3^6, 42)$
$x_3^7(0, 14) = x_0^7(0, 14) \oplus ROTL(x_1^6, 11)$

$x_0^8(0) = x_0^7(0, 8) \boxplus_w x_1^7(0, 8) \boxplus_w k_2(0, 8)$
$x_2^8(0) = x_2^7(0) \boxplus_w x_3^7(0) \boxplus_w k_4(0)$
$x_1^8(0) = (x_2^8(0) \oplus ROTL(x_3^7, 50)) \boxplus_w k_3(0)$
$x_3^8(0, 8) = (x_0^8(0, 8) \oplus ROTL(x_1^7, 59)) \boxplus_w k_0(0) \boxplus_w 2$
$x_0^9(0) = x_0^8(0) \boxplus_w x_1^8(0)$
$x_2^9(0) = x_2^8(0) \boxplus_w x_3^8(0)$
$x_1^9(0) = x_2^9(0) \oplus ROTL(x_3^8(8), 56)$
$x_0^{10}(0) = x_0^9(0) \boxplus_w x_1^9(0)$

$win_0^{12} = \{0, 7, 13, 58\}$
$d_0^{12}(win_0^{12})^* = d_0^{12}(win_0^{12}) \boxminus_w k_3(win_0^{12})$
$d_1^{12}(57)^* = d_1^{12}(57) \boxminus_w k_4(57)$
$d_2^{12}(57)^* = d_2^{12}(57) \boxminus_w k_0(57)$
$d_3^{12}(7, 58)^* = d_3^{12}(7, 58) \boxminus_w k_1(7, 58)$
$d_0^{11}(0, 13) = d_0^{12}(0, 13)^* \boxminus_w (d_0^{12}(58, 7)^* \oplus d_3^{12}(58, 7)^*)$
$d_3^{11}(13) = d_2^{12}(57)^* \oplus d_1^{12}(58)^*$

$d_0^{10}(0) = d_0^{11} \boxminus_w (d_0^{11}(13) \oplus d_3^{11}(13)$