The Additive Differential Probability of ARX[⋆]

Vesselin Velichkov[1,2,⋆⋆], Nicky Mouha[1,2,⋆⋆⋆],
Christophe De Cannière[1,2,†], and Bart Preneel[1,2]

[1] Department of Electrical Engineering ESAT/SCD-COSIC,
Katholieke Universiteit Leuven. Kasteelpark Arenberg 10, B-3001 Heverlee, Belgium
[2] Interdisciplinary Institute for BroadBand Technology (IBBT), Belgium
{Vesselin.Velichkov,Nicky.Mouha,Christophe.DeCanniere}@esat.kuleuven.be

Abstract. We analyze $\mathrm{adp}^{\mathtt{ARX}}$, the probability with which additive differences propagate through the following sequence of operations: modular addition, bit rotation and XOR (ARX). We propose an algorithm to evaluate $\mathrm{adp}^{\mathtt{ARX}}$ with a linear time complexity in the word size. This algorithm is based on the recently proposed concept of S-functions. Because of the bit rotation operation, it was necessary to extend the S-functions framework. We show that $\mathrm{adp}^{\mathtt{ARX}}$ can differ significantly from the multiplication of the differential probability of each component. To the best of our knowledge, this paper is the first to propose an efficient algorithm to calculate $\mathrm{adp}^{\mathtt{ARX}}$. Accurate calculations of differential probabilities are necessary to evaluate the resistance of cryptographic primitives against differential cryptanalysis. Our method can be applied to find more accurate differential characteristics for ARX-based constructions.

Keywords: Additive differential probability, differential cryptanalysis, symmetric-key, ARX.

1 Introduction

Many cryptographic primitives are built using the operations modular addition, bit rotation and XOR (ARX). The advantage of using these operations is that they are very fast when implemented in software. At the same time, they have desirable cryptographic properties. Modular addition provides non-linearity, bit rotation provides diffusion within a single word, and XOR provides diffusion between words and linearity. A disadvantage of using these operations is that the diffusion is typically slow. This is often compensated for by adding more rounds to the designed primitive.

[⋆] This work was supported in part by the Research Council K.U.Leuven: GOA TENSE, and by the IAP Program P6/26 BCRYPT of the Belgian State (Belgian Science Policy), and in part by the European Commission through the ICT program under contract ICT-2007-216676 ECRYPT II.

[⋆⋆] DBOF Doctoral Fellow, K.U.Leuven, Belgium.

[⋆⋆⋆] This author is funded by a research grant of the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen).

[†] Postdoctoral Fellow of the Research Foundation – Flanders (FWO).

Examples of cryptographic algorithms that make use of the addition, `XOR` and rotate operations, are the stream ciphers Salsa20 [2] and HC-128 [16], the block cipher XTEA [13], the MD4-family of hash functions (including MD5 and SHA-1), as well as 6 out of the 14 candidates of NIST's SHA-3 hash function competition [12]: BLAKE [1], Blue Midnight Wish [7], CubeHash [3], Shabal [4], SIMD [8] and Skein [6].

Differential cryptanalysis is one of the main techniques to analyze cryptographic primitives. Therefore, it is essential that the differential properties of `ARX` are well understood both by designers and attackers. Several important results have been published in this direction. In [15], Meier and Staffelbach present the first analysis of the propagation of the carry bit in modular addition. Later, Lipmaa and Moriai proposed an algorithm to compute the `XOR` differential probability of modular addition (xdp^{+}) [9]. Its dual, the additive differential probability of `XOR` ($\mathrm{adp}^{\oplus}$), was analyzed by Lipmaa, Wallén and Dumas in [10]. The latter proposed new algorithms for the computation of both xdp^{+} and $\mathrm{adp}^{\oplus}$, based on matrix multiplications. The differential properties of bit rotation have been analyzed by Daum in [5].

In [11], Mouha et al. propose the concept of S-functions. S-functions are a class of functions that can be computed bitwise, so that the i-th output bit is computed using only the i-th input bits and a finite state $S[i]$. Although S-functions have been analyzed before, [11] is the first paper to present a fully generic and efficient framework to determine their differential properties. The methods used in the proposed framework are based on graph theory, and the calculations can be efficiently performed using matrix multiplications.

In this paper, we extend the S-function framework to compute the differential probability $\mathrm{adp}^{\mathtt{ARX}}$ of the following sequence of operations: addition, bit rotation and `XOR`. We describe a method to compute $\mathrm{adp}^{\mathtt{ARX}}$ based on the matrix multiplication technique proposed in [10], and generalized in [11]. The time complexity of our algorithm is linear in the word size. We provide a formal proof of its correctness, and also confirm it experimentally. We performed experiments on all combinations of 4-bit inputs and on a number of random 32-bit inputs.

We observe that $\mathrm{adp}^{\mathtt{ARX}}$ can differ significantly from the probability obtained by multiplying the differential probabilities of addition, rotation and `XOR`. This confirms the need for an efficient calculation of the differential probability for the `ARX` operation. We are unaware of any results in existing literature where $\mathrm{adp}^{\mathtt{ARX}}$ is calculated efficiently. Accurate and efficient calculations of differential probabilities are required for the efficient search for characteristics used in differential cryptanalysis.

The outline of the paper is as follows. In Sect. 2, we define the additive differential probability of bit rotation ($\mathrm{adp}^{\lll}$). We give an overview of S-functions and we describe how they can be used to compute the additive differential probability of `XOR` ($\mathrm{adp}^{\oplus}$) in Sect. 3. The additive differential probability of `ARX` ($\mathrm{adp}^{\mathtt{ARX}}$) is defined in Sect. 4. We show that $\mathrm{adp}^{\mathtt{ARX}}$ can deviate significantly from the product of the probabilities of rotation and `XOR`. In Sect. 5, we propose a method for the calculation of $\mathrm{adp}^{\mathtt{ARX}}$. The theorem stating its correctness is formulated in

Table 1. Notation

Symbol	**Meaning**
n	Number of bits in one word
x	n-bit word
$x[i]$	Select the $(i \mod n)$-th bit (or element) of the n-bit word x, $x[0]$ is the least-significant bit (or element)
$+$	Addition modulo 2^n
-	Subtraction modulo 2^n
r	Rotation constant, $0 \leq r < n$
$\lll r$	Left bit rotation by r positions
$\ggg r$	Right bit rotation by r positions
$\gg 1$	A signed shift by one position to the right (e.g. $-1 \gg 1 = -1$)
$\oplus$	Exclusive-OR (XOR)
Δx	n-bit additive difference $(x_2 - x_1) \mod 2^n$
$\|\|$	Concatenation of bit strings
ARX	The sequence of the operations: $+, \lll, \oplus$
$\mathrm{adp}^{\lll}$	The additive differential probability of bit rotation
$\mathrm{adp}^{\oplus}$	The additive differential probability of XOR
$\mathrm{adp}^{\mathtt{ARX}}$	The additive differential probability of ARX
x_2	Number x in binary representation
$\Delta\alpha \to \Delta\beta$	Input difference $\Delta\alpha$ propagates to output difference $\Delta\beta$

Sect. 6. In Sect. 7, we confirm the computation of $\mathrm{adp}^{\mathtt{ARX}}$ experimentally. Section 8 concludes the paper. The matrices used to compute $\mathrm{adp}^{\mathtt{ARX}}$ are given in Appendix A. Appendix B contains the full proof of correctness of the $\mathrm{adp}^{\mathtt{ARX}}$ algorithm. Throughout the paper, we use the notation listed in Table 1.

2 Definition of $\mathrm{adp}^{\lll}$

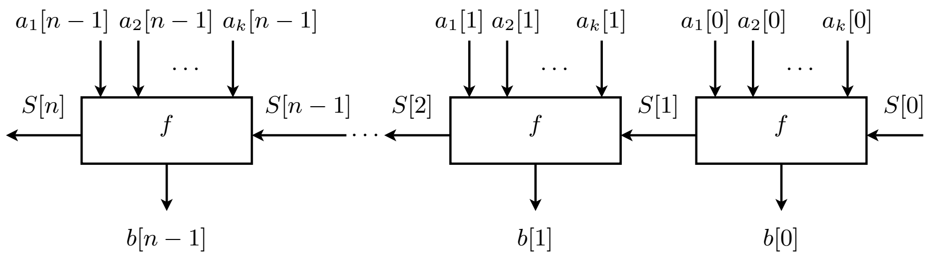
The additive differential probability of bit rotation, denoted by $\mathrm{adp}^{\lll}$, is the probability with which additive differences propagate through bit rotation. This probability was studied by Daum in [5]. We give a brief summary of the results in [5] that are relevant to our work.

Let $\Delta\alpha$ be a fixed additive difference. Let a_1 be an n-bit word chosen uniformly at random and $(a_1, a_1 + \Delta\alpha)$ be a pair of n-bit words input to a left rotation by r positions. Let $\Delta\beta$ be the output additive difference between the rotated inputs:

$$\Delta\beta = ((a_1 + \Delta\alpha) \lll r) - (a_1 \lll r) \ . \tag{1}$$

In [5, Corollary 4.14, Case 2] it is shown that there are four possibilities for $\Delta\beta$:

$$\Delta\beta \in \{\Delta\beta_{u,v} = (\Delta\alpha \lll r) - u2^r + v, \quad u, v \in \{0, 1\}\} \ . \tag{2}$$



$a_1[n-1]$ $a_2[n-1]$ $a_k[n-1]$
$a_1[1]$ $a_2[1]$ $a_k[1]$
$a_1[0]$ $a_2[0]$ $a_k[0]$
$S[n]$
$S[n-1]$
$S[2]$
$S[1]$
$S[0]$
f
$b[n-1]$
$b[1]$
$b[0]$

Fig. 1. Representation of an S-function

The probabilities for the output differences $\Delta\beta$ are:

$$P_{0,0} = P(\Delta\alpha \to \Delta\beta_{0,0}) = 2^{-n}(2^r - \Delta\alpha_L)(2^{n-r} - \Delta\alpha_R) \ , \tag{3}$$

$$P_{0,1} = P(\Delta\alpha \to \Delta\beta_{0,1}) = 2^{-n}(2^r - \Delta\alpha_L - 1)\Delta\alpha_R \ , \tag{4}$$

$$P_{1,0} = P(\Delta\alpha \to \Delta\beta_{1,0}) = 2^{-n}\Delta\alpha_L(2^{n-r} - \Delta\alpha_R) \ , \tag{5}$$

$$P_{1,1} = P(\Delta\alpha \to \Delta\beta_{1,1}) = 2^{-n}(\Delta\alpha_L + 1)\Delta\alpha_R \ . \tag{6}$$

In the above equations, $\Delta\alpha_L$ is the word composed of the r most significant bits of $\Delta\alpha$ and $\Delta\alpha_R$ is the word composed of the $n-r$ least significant bits of $\Delta\alpha$ such that

$$\Delta\alpha = \Delta\alpha_L \parallel \Delta\alpha_R \ . \tag{7}$$

We define the additive differential probability of bit rotation as

$$\mathrm{adp}^{\lll}(\Delta\alpha \xrightarrow{r} \Delta\beta) = \begin{cases} P_{u,v} & , \text{ if } \Delta\beta = \Delta\beta_{u,v} \text{ for some } u, v \in \{0,1\} \ , \\ 0 & , \text{ otherwise } . \end{cases} \tag{8}$$

3 Computation of adp$^{\oplus}$ Using S-Functions

S-functions were introduced by Mouha et al. in [11]. An S-function (short for *state*-function) accepts n-bit words $a_1, a_2, \ldots, a_k$ and a list of states $S[i]$ (for $0 \le i < n$) as input, and produces an n-bit output word b in the following way:

$$(b[i], S[i+1]) = f(a_1[i], a_2[i], \ldots, a_k[i], S[i]), \quad 0 \le i < n \ . \tag{9}$$

Initially, we set $S[0] = 0$. A schematic representation of an S-function is given in Fig. 1.

In [11], S-functions were used to compute the additive differential probability of XOR (adp$^{\oplus}$). This is the probability with which additive differences propagate through the XOR operation. The results of [11] confirm the calculation of adp$^{\oplus}$ obtained in [10]. They are relevant to the calculation of adp$^{\mathtt{ARX}}$, and will therefore be briefly described below.

Fix the additive differences $\Delta\alpha, \Delta\beta, \Delta\gamma$. With Δe we designate the additive difference:

$$\Delta e = e_2 - e_1 = ((c_1 + \Delta\alpha) \oplus (d_1 + \Delta\beta)) - (c_1 \oplus d_1) \ . \tag{10}$$

The probability $\mathrm{adp}^{\oplus}$ is equal to the number of pairs (c_1, d_1) for which $\Delta e = \Delta\gamma$, divided by the total number of pairs (c_1, d_1):

$$\mathrm{adp}^{\oplus}(\Delta\alpha, \Delta\beta \to \Delta\gamma) = \frac{|\{(c_1, d_1) : \Delta e = \Delta\gamma\}|}{|\{(c_1, d_1)\}|} \ . \tag{11}$$

The i-th bit of the output difference $\Delta e[i]$ can be computed from the i-th bits of the input differences $\Delta\alpha[i], \Delta\beta[i]$ and the state $S[i]$. The state $S[i]$ consists of the carries $s_1[i], s_2[i]$ and the borrow $s_3[i]$:

$$s_1[i] = (c_1[i-1] + \Delta\alpha[i-1] + s_1[i-1]) \gg 1 \ , \tag{12}$$
$$s_2[i] = (d_1[i-1] + \Delta\beta[i-1] + s_2[i-1]) \gg 1 \ , \tag{13}$$
$$s_3[i] = (e_2[i-1] - e_1[i-1] + s_3[i-1]) \gg 1 \ , \tag{14}$$

where $s_1[0] = s_2[0] = s_3[0] = 0$. Note that the bit shift by one position to the right in (14) is a signed shift (e.g. $-1 \gg 1 = -1$). The S-function for $\mathrm{adp}^{\oplus}$ is defined as

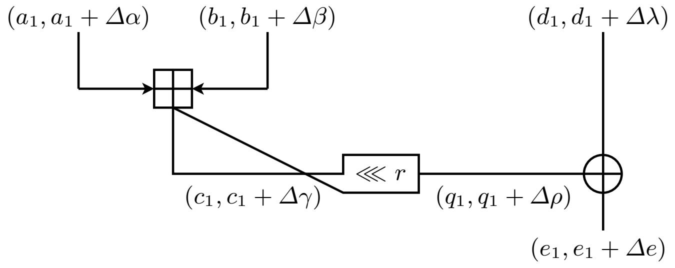
$$(\Delta e[i], S[i+1]) = f(c_1[i], d_1[i], \Delta\alpha[i], \Delta\beta[i], S[i]), \quad 0 \le i < n \ . \tag{15}$$

By definition, the state $S[i]$ of an S-function has the same fixed size for every $0 \le i < n$. In the case of $\mathrm{adp}^{\oplus}$, this size is 3 bits. Therefore, there are eight distinct states $S[i]$ in total for any bit position $0 \le i < n$. For fixed input differences, the transition between consecutive states $S[i]$ and $S[i+1]$ can be described by an 8×8 adjacency matrix. There are eight such matrices in total – one for each value of the 3-tuple $(\Delta\alpha[i], \Delta\beta[i], \Delta\gamma[i])$. These eight matrices are derived in [11] and are shown to be equal (up to a permutation) to the matrices previously computed in [10].

The probability $\mathrm{adp}^{\oplus}$ is computed by iterating over all bit positions 0 through $n-1$. At each position i, one of the eight matrices is selected depending on the value of the bits of the differences $\Delta\alpha[i], \Delta\beta[i], \Delta\gamma[i]$. All n matrices that are selected in this way are multiplied. The resulting matrix is right-multiplied by the column vector representing the initial state. We now obtain a column vector. After summing its elements, we end up with the number of pairs (a_1, b_1) for which $\Delta e = \Delta\gamma$. The probability $\mathrm{adp}^{\oplus}$ is computed by dividing the obtained value by the total number of pairs (a_1, b_1). This whole process is summarized by the following formula:

$$\mathrm{adp}^{\oplus}(\Delta\alpha, \Delta\beta \to \Delta\gamma) = 2^{-2n} L A_{w[n-1]} \cdots A_{w[1]} A_{w[0]} C \ . \tag{16}$$

In (16), the factor 2^{2n} corresponds to the total number of n-bit pairs (c_1, d_1). $C = \begin{pmatrix} 1\,0\,0\,0\,0\,0\,0\,0 \end{pmatrix}^T$ is a column vector indicating the initial state $S[0] = 0$,



$(a_1, a_1 + \Delta\alpha)$
$(b_1, b_1 + \Delta\beta)$
$(d_1, d_1 + \Delta\lambda)$
$\lll r$
$(c_1, c_1 + \Delta\gamma)$
$(q_1, q_1 + \Delta\rho)$
$(e_1, e_1 + \Delta e)$

Fig. 2. Additive differences passing through the `ARX` operation

corresponding to the two initial carries $s_1[0], s_2[0]$ and the initial borrow $s_3[0]$ being equal to zero. Multiplication by the row vector $L = \begin{pmatrix}1 & 1 & 1 & 1 & 1 & 1 & 1 & 1\end{pmatrix}$ is equivalent to adding the elements of the column vector resulting from the product $A_{w[n-1]} \cdots A_{w[0]} C$. The matrix indices $w[i], 0 \leq i < n$ are in the set $\{0, 1, \ldots, 7\}$. Index $w[i]$ is obtained by concatenating the i-th bits of the differences: $w[i] = \Delta\alpha[i] \parallel \Delta\beta[i] \parallel \Delta\gamma[i]$. At every bit position i, the index $w[i]$ selects one of the eight distinct adjacency matrices $A_{w[i]}$. They are given in Appendix A.

In the next sections, we define the additive differential probability of `ARX` and we describe a method to compute this probability using S-functions.

4 Definition of $\text{adp}^{\texttt{ARX}}$

The operation `ARX` is defined as:

$$\texttt{ARX}(a, b, d, r) \overset{\text{def}}{=} ((a + b) \lll r) \oplus d \ . \tag{17}$$

Let the additive differences $\Delta\alpha, \Delta\beta, \Delta\lambda, \Delta\eta$ be fixed. Let Δe be the difference between two outputs of `ARX`:

$$\Delta e = e_2 - e_1 = \texttt{ARX}(a_1 + \Delta\alpha, b_1 + \Delta\beta, d_1 + \Delta\lambda, r) - \texttt{ARX}(a_1, b_1, d_1, r) \ . \tag{18}$$

Equation (18) is illustrated in Fig. 2. Additive differences pass through modular addition with probability one. Therefore we can directly compute the output difference after the addition: $\Delta\gamma = \Delta\alpha + \Delta\beta$. Let $c_1 = a_1 + b_1$ be any output from the addition (Fig. 2). The additive differential probability of `ARX` is defined as the number of pairs (c_1, d_1) for which $\Delta e = \Delta\eta$, divided by all pairs (c_1, d_1):

$$\text{adp}^{\texttt{ARX}}(\Delta\gamma, \Delta\lambda \xrightarrow{r} \Delta\eta) \overset{\text{def}}{=} \frac{|\{(c_1, d_1) : \Delta e = \Delta\eta\}|}{|\{(c_1, d_1)\}|} \ . \tag{19}$$

An estimation of $\text{adp}^{\texttt{ARX}}$ can be obtained as the product of the probabilities of rotation and `XOR`. We designate the probability computed in this way by $P_{\texttt{rotxor}}$:

$$P_{\texttt{rotxor}} = \sum_{j=0}^{4} (\text{adp}^{\lll}(\Delta\gamma \xrightarrow{r} \Delta\rho_j) \cdot \text{adp}^{\oplus}(\Delta\rho_j, \Delta\lambda \to \Delta\eta)) \ , \tag{20}$$

where $\Delta\rho_j, 0 \leq j < 4$ are the four possible output differences after the rotation (2). Equation (20) would be an accurate evaluation of $\mathrm{adp}^{\mathtt{ARX}}$ if the inputs to the rotation and the inputs to the XOR operation were independent. In reality they are not, as illustrated by the following example.

Example 1. Let $n = 4$, $r = 1$, $\Delta\gamma = \mathtt{1000}_2$, $\Delta\lambda = \mathtt{0000}_2$, $\Delta\eta = \mathtt{0001}_2$. Two output differences after the rotation are possible: $\Delta\rho_0 = \mathtt{0001}_2$ and $\Delta\rho_2 = \mathtt{1111}_2$, each with probability 2^{-1}. They both propagate through the XOR operation with probability $2^{-1.54}$. The total probability $P_{\mathtt{rotxor}}$ is

$$\begin{aligned} P_{\mathtt{rotxor}} &= \mathrm{adp}^{\lll}(\mathtt{1000}_2 \xrightarrow{1} \mathtt{0001}_2) \cdot \mathrm{adp}^{\oplus}(\mathtt{0001}_2, \mathtt{0000}_2 \rightarrow \mathtt{0001}_2) \\ &+ \mathrm{adp}^{\lll}(\mathtt{1000}_2 \xrightarrow{1} \mathtt{1111}_2) \cdot \mathrm{adp}^{\oplus}(\mathtt{1111}_2, \mathtt{0000}_2 \rightarrow \mathtt{0001}_2) \\ &= 2^{-1} \cdot 2^{-1.54} + 2^{-1} \cdot 2^{-1.54} = 2^{-1.54} \ . \end{aligned} \tag{21}$$

The actual probability is, however, higher than $P_{\mathtt{rotxor}}$ and is $P_{\mathrm{exper}} = 2^{-1}$. The reason for the discrepancy is the fact that there exist pairs of inputs to XOR that satisfy the differences $\Delta\rho_0$ or $\Delta\rho_2$, but when they are rotated back they do not satisfy the difference $\Delta\gamma$. One such input pair is $(q_1, q_2) = (2, 1)$. This pair satisfies the difference $\Delta\rho_2$: $q_2 - q_1 = (1 - 2) \mod 16 = 15 = \mathtt{1111}_2$. Yet, it does not satisfy the difference $\Delta\gamma$: $(q_2 \ggg 1) - (q_1 \ggg 1) = (8 - 1) \mod 16 = 7 = \mathtt{0111}_2 \neq \mathtt{1000}_2$. There are 8 such pairs in total: $(0, 15)$, $(2, 1)$, $(4, 3)$, $(6, 5)$, $(8, 7)$, $(10, 9)$, $(12, 11)$, $(14, 13)$. Given the output difference $\Delta\rho_2$, these pairs are impossible. Thus the total number of possible inputs to the XOR is reduced from 256 to 128. The reason is, that for every impossible pair (q_1, q_2), there are 16 possibilities for the second input pair $(d_1 + \Delta\lambda, d_1)$. Of those 128 pairs, 64 satisfy the output difference $\Delta\eta$. Thus the actual probability is $\mathrm{adp}^{\oplus}(\mathtt{1111}_2, \mathtt{0000}_2 \rightarrow \mathtt{0001}_2) = 64/128 = 2^{-1}$ and not $88/256 = 2^{-1.54}$. We have a similar situation for the difference $\Delta\rho_0$. In that case the impossible pairs are $(1, 2)$, $(3, 4)$, $(5, 6)$, $(7, 8)$, $(9, 10)$, $(11, 12)$, $(13, 14)$, $(15, 0)$ and the $\mathrm{adp}^{\oplus}$ probability is again 2^{-1}. Thus the final probability $\mathrm{adp}^{\mathtt{ARX}}$ is 2^{-1}.

5 Computation of $\mathrm{adp}^{\mathtt{ARX}}$

In Example 1, we showed that the inputs to the rotation and to the XOR operation are not independent. This causes the additive differential probability of ARX, estimated by the multiplication of the probabilities of the rotation and the XOR, to differ from the actual probability. This problem can be solved if the intermediate differences $\Delta\rho_j, 0 \leq j < 4$ are not computed explicitly. Consider the ARX operation (17). Let $a_1 + b_1 = c_1$, $q_1 = (c_1 \lll r)$ and $e_1 = \mathtt{ARX}(a_1, b_1, d_1, r), e_2 = \mathtt{ARX}(a_2, b_2, d_2, r)$, as shown in Fig. 2. Note that $c_1[i] = q_1[i + r]$. Therefore $q_1[i + r] \oplus d_1[i + r] = e_1[i + r]$ is equivalent to

$$c_1[i] \oplus d_1[i + r] = e_1[i + r] \ . \tag{22}$$

Using this representation, we can compute the bits of the output e_1 without using the intermediate variable q_1. Consequently, we can compute the output difference $\Delta e = e_2 - e_1$ without using the intermediate differences $\Delta\rho_i$:

$$c_2[i] = c_1[i] \oplus \Delta\gamma[i] \oplus s_1[i] \ , \tag{23}$$
$$d_2[i+r] = d_1[i+r] \oplus \Delta\lambda[i+r] \oplus s_2[i+r] \ , \tag{24}$$
$$\Delta e[i+r] = e_1[i+r] \oplus e_2[i+r] \oplus s_3[i+r] \ , \tag{25}$$

where

$$s_1[i] = (c_1[i-1] + \Delta\gamma[i-1] + s_1[i-1]) \gg 1 \ , \tag{26}$$
$$s_2[i+r] = (d_1[i+r-1] + \Delta\lambda[i+r-1] + s_2[i+r-1]) \gg 1 \ , \tag{27}$$
$$s_3[i+r] = (e_2[i+r-1] - e_1[i+r-1] + s_3[i+r-1]) \gg 1 \ . \tag{28}$$

The S-function for $\mathrm{adp}^{\mathtt{ARX}}$ is defined as

$$(\Delta e[i+r], S[i+1]) = f(c_1[i], d_1[i+r], \Delta\gamma[i], \Delta\lambda[i+r], S[i]), \quad 0 \le i < n \ . \tag{29}$$

The definitions of the S-functions for $\mathrm{adp}^{\mathtt{ARX}}$ (29) and $\mathrm{adp}^{\oplus}$ (15) are very similar. Yet the computation of the two differ in several aspects. We describe these differences below.

5.1 The Initial State

As described in Sect. 3 for $\mathrm{adp}^{\oplus}$, the state is composed of two carries and one borrow arising from the three modular operations involved in computing the output (10). At position $i = 0$, these values are all zero. Therefore, the initial state is $S[0] = (s_1[0], s_2[0], s_3[0]) = (0, 0, 0)$. In the case of $\mathrm{adp}^{\mathtt{ARX}}$ the situation is slightly different. The reason is that when we perform the `ARX` operation bitwise, at position 0, we compute the 0-th bit of c_2 and the r-th bits of d_2 and Δe (23)-(25). Similarly to $\mathrm{adp}^{\oplus}$, the carry $s_1[0]$ is zero. However the carry $s_2[r]$ and the borrow $s_3[r]$ are not necessarily zero:

$$s_1[0] = 0 \ , \tag{30}$$
$$s_2[r] = (d_1[r-1] + \Delta\lambda[r-1] + s_2[r-1]) \gg 1 \ , \tag{31}$$
$$s_3[r] = (e_2[r-1] - e_1[r-1] + s_3[r-1]) \gg 1 \ . \tag{32}$$

Thus the initial state of the $\mathrm{adp}^{\mathtt{ARX}}$ S-function is $S[0] = (s_1[0], s_2[r], s_3[r])$. Because $s_2[r] \in \{0, 1\}$ and $s_3[r] \in \{-1, 0\}$, there are four possibilities for $S[0]$. Each of them corresponds to one of the 3-tuples $(0, 0, -1)$, $(0, 1, -1)$, $(0, 0, 0)$, $(0, 1, 0)$. We map all 8 possible values of any state $S[i] = (s_1[i], s_2[i+r], s_3[i+r])$ to the set of integers $\{0, 1, ..., 7\}$ as shown in Table 2. Following this convention, $S[0] \in \{0, 2, 4, 6\}$.

5.2 The Final State

From (30)-(32), it follows that in order to compute $S[0]$ we have to know $s_2[r-1]$ and $s_3[r-1]$. In other words, in order to compute the initial state of the $\mathrm{adp}^{\mathtt{ARX}}$

Table 2. Mapping between the 8 states of the $\mathtt{adp}^{\mathtt{ARX}}$ S-function and the set of integers $\{0, \ldots, 7\}$

$\mathbf{S[i]}$	0	1	2	3	4	5	6	7
$(\mathbf{s_1[i], s_2[i+r], s_3[i+r]})$	(0,0,-1)	(1,0,-1)	(0,1,-1)	(1,1,-1)	(0,0,0)	(1,0,0)	(0,1,0)	(1,1,0)

S-function we need information from the final state $S[n-1] = (s_1[n-1], s_2[r-1], s_3[r-1])$. However, at the start of the computation ($i = 0$) we do not know the output of the S-function at position $i = n-1$ yet. We solve this problem by iterating over all four values of $(s_2[r-1], s_3[r-1])$ at $i = 0$. For each of them, we compute $S[0]$ and we proceed with the computation of the S-function. From the set of final output states $S[n-1]$, we accept as valid only those that match the values of $(s_2[r-1], s_3[r-1])$ at position $i = 0$. Each value of the tuple $(s_2[r-1], s_3[r-1])$ will match exactly two of all eight final states $S[n-1]$ corresponding to the two possibilities for $c_1[n-1] \in \{0, 1\}$. For example the initial state $(0, 0, -1)$ will be matched by final states $(0, 0, -1)$ and $(1, 0, -1)$. In general, following the mapping in Table 2, an initial state $S[0] = j \in \{0, 2, 4, 6\}$ will match final states $S[n-1] = j$ and $S[n-1] = j + 1$.

5.3 A Special Intermediate State

There is one final issue that should be taken care of, before we are able to compute $\mathtt{adp}^{\mathtt{ARX}}$. Consider step $i = n - r - 1$ of the computation of the S-function of $\mathtt{adp}^{\mathtt{ARX}}$. At this step, we are operating on bits at position $n - 1$ in order to compute $s_2[0]$ and $s_3[0]$. Since these are the most-significant input bits, the carries and borrows that they generate should be discarded. Consequently, $s_2[0]$ and $s_3[0]$ should be set to zero at this step:

$$s_1[n-r] = (c_1[n-r-1] + \Delta\gamma[n-r-1] + s_1[n-r-1]) \gg 1 \ , \tag{33}$$

$$s_2[0] = 0 \ , \tag{34}$$

$$s_3[0] = 0 \ . \tag{35}$$

Therefore state $S[n-r] = (s_1[n-r], s_2[0], s_3[0])$ is a special intermediate state for which the only permissible values are $(0, 0, 0)$ and $(1, 0, 0)$ i.e. $S[n-r] \in \{4, 5\}$. Because of this special state, it is necessary to construct an 8×8 projection matrix R in addition to the matrices A_q, $0 \leq q < 8$ used in the computation of $\mathtt{adp}^{\oplus}$ (16). By multiplying the matrix $A_{w[n-r-1]}$ at position $n - r - 1$ to the left by R, the transition from the set of output states corresponding to the value of the 3-tuple $(\Delta\gamma[n-r], \Delta\lambda[0], \Delta\eta[0])$ to the set of reachable output states is performed (cf. Sect. 3). This operation effectively transforms every state $S[n-r] = (s_1[n-r], s_2[0], s_3[0])$ to the permissible value for the special state $S[n-r] = (s_1[n-r], 0, 0)$.

5.4 Computing $\mathrm{adp}^{\mathtt{ARX}}$

The probability $\mathrm{adp}^{\mathtt{ARX}}$ can be computed as follows:

$$\mathrm{adp}^{\mathtt{ARX}}(\Delta\gamma, \Delta\lambda \xrightarrow{r} \Delta\eta) = \\ 2^{-2n} \sum_{j\in\{0,2,4,6\}} (L_j A_{w[n-1]} \cdots A_{w[n-r]} R A_{w[n-r-1]} \cdots A_{w[1]} A_{w[0]} C_j) \ . \qquad (36)$$

In (36), $j \in \{0, 2, 4, 6\}$ iterates over the four possible initial states. The binary column vector C_j of dimension 8×1 indicates the initial state. It has 1 at position j and 0 elsewhere. The vector L_j is a 1×8 binary row vector that has 1 at positions j and $j+1$ and has 0 elsewhere. By multiplying the result of the matrix multiplication by L_j, we are effectively adding only the two final states that correspond to the initial state j (cf. Sect. 5.2). The indices $w[0], \ldots, w[n-1]$ are in the set $\{0, 1, \ldots, 7\}$. Index $w[i]$ is obtained by concatenating the corresponding bits of the differences: $w[i] = \Delta\gamma[i] \parallel \Delta\lambda[i + r] \parallel \Delta\eta[i + r]$. For every bit position $0 \le i < n$, index $w[i]$ selects one of the eight 8×8 adjacency matrices A_q, $0 \le q < 8$. For position $i = n - r - 1$, matrix $A_{w[n-r-1]}$ is additionally multiplied to the left by the projection matrix R. The matrices A_q are the same as the the ones used in the computation of $\mathrm{adp}^{\oplus}$. Matrices A_q and R are given in Appendix A.

The computation of $\mathrm{adp}^{\mathtt{ARX}}$ (36) is slightly different from the computation of $\mathrm{adp}^{\oplus}$ (16). The main difference is that there are four evaluations of the S-function. From each of them, two of the eight final states are selected. The second difference is the presence of the additional projection matrix R.

In Example 2, we demonstrate the computation of $\mathrm{adp}^{\mathtt{ARX}}$ for the additive differences given in Example 1.

Example 2. For $n = 4$, $r = 1$, $\Delta\gamma = \mathtt{1000}_2$, $\Delta\lambda = \mathtt{0000}_2$, $\Delta\eta = \mathtt{0001}_2$, we want to compute $\mathrm{adp}^{\mathtt{ARX}}(\Delta\gamma, \Delta\lambda \xrightarrow{r} \Delta\eta)$. First we compute the indices $w[i] = \Delta\gamma[i] \parallel \Delta\lambda[i + 1] \parallel \Delta\eta[i + 1]$, $0 \le i < 4$:

$$\begin{aligned} w[0] &= \Delta\gamma[0] \parallel \Delta\lambda[1] \parallel \Delta\eta[1] = 000 \ , \\ w[1] &= \Delta\gamma[1] \parallel \Delta\lambda[2] \parallel \Delta\eta[2] = 000 \ , \\ w[2] &= \Delta\gamma[2] \parallel \Delta\lambda[3] \parallel \Delta\eta[3] = 000 \ , \\ w[3] &= \Delta\gamma[3] \parallel \Delta\lambda[0] \parallel \Delta\eta[0] = 101 \ . \end{aligned}$$

Indices $w[0], w[1], w[2]$ select matrix A_{000}; index $w[3]$ selects matrix A_{101}. The probability $\mathrm{adp}^{\mathtt{ARX}}$ is computed as

$$\begin{aligned} &\mathrm{adp}^{\mathtt{ARX}}(\mathtt{1000}_2, \mathtt{0000}_2 \xrightarrow{1} \mathtt{0001}_2) \\ &= 2^{-8} \sum_{j\in\{0,2,4,6\}} L_j A_{101} R A_{000} A_{000} A_{000} C_j = 2^{-1} \ , \end{aligned}$$

where

$$\begin{aligned}
C_0 &= \begin{pmatrix}1\,0\,0\,0\,0\,0\,0\,0\end{pmatrix}^T ,\; L_0 = \begin{pmatrix}1\,1\,0\,0\,0\,0\,0\,0\end{pmatrix} \;, \\
C_2 &= \begin{pmatrix}0\,0\,1\,0\,0\,0\,0\,0\end{pmatrix}^T ,\; L_2 = \begin{pmatrix}0\,0\,1\,1\,0\,0\,0\,0\end{pmatrix} \;, \\
C_4 &= \begin{pmatrix}0\,0\,0\,0\,1\,0\,0\,0\end{pmatrix}^T ,\; L_4 = \begin{pmatrix}0\,0\,0\,0\,1\,1\,0\,0\end{pmatrix} \;, \\
C_6 &= \begin{pmatrix}0\,0\,0\,0\,0\,0\,1\,0\end{pmatrix}^T ,\; L_6 = \begin{pmatrix}0\,0\,0\,0\,0\,0\,1\,1\end{pmatrix} \;.
\end{aligned}$$

6 Proof of Correctness

With the following theorem we state that the computation of the probability $\mathrm{adp}^{\mathtt{ARX}}$ (36) is correct. To prove this, we use arguments similar to [11, §3.4, Theorem 1]. The full proof is given in Appendix B. In this section, we provide the intuition behind it.

Theorem 1.

$$2^{-2n} \sum_{j\in\{0,2,4,6\}} (L_j A_{w[n-1]} \cdots A_{w[n-r]} R A_{w[n-r-1]} \cdots A_{w[1]} A_{w[0]} C_j) = \frac{|\{(c_1, d_1) : \Delta e = \Delta\eta\}|}{|\{(c_1, d_1)\}|} \;. \tag{37}$$

Proof. The full proof is given in Appendix B.

Theorem 1 states that the probability computed using the proposed method (36) is equal to the probability $\mathrm{adp}^{\mathtt{ARX}}$ as defined by (19). The most natural way to see this is by using a graph representation of the S-function [11]. In [11, §3.4] it is shown that an S-function can be represented as a directed acyclic graph composed of bipartite subgraphs. In this representation, all pairs of inputs that satisfy the input differences are equal to the number of paths through the graph. Each path connects a valid initial state to a valid final state. A subset of these paths corresponds to the set of input pairs that satisfy both the input and the output differences. Therefore, the computation of the S-function is equivalent to counting the number of paths in the subset and dividing the result by the number of all paths. Since, in the process, no path is counted more than once, the result is exactly equal to $\mathrm{adp}^{\mathtt{ARX}}$ as defined by (19).

7 Experiments

In this section, we confirm the correctness of the computation of $\mathrm{adp}^{\mathtt{ARX}}$ (36) experimentally. We performed two sets of experiments: one for 4-bit words and one for 32-bit words. In both sets, we compare three computations of the additive differential probability of ARX:

– $P_{\mathtt{exper}}$: the probability computed experimentally, using (19), over a certain number of inputs that satisfy the input differences

Table 3. Comparing three ways of computing the additive differential probability of $\mathtt{ARX}$ for 4-bit words. Shown are the 24 cases for which the deviation of $P_{\mathtt{rotxor}}$ (20) from the experimentally obtained value $P_{\mathtt{exper}}$ (19) is highest: $|P_{\mathtt{exper}} - P_{\mathtt{rotxor}}| > 0.1$. The probability $\mathrm{adp}^{\mathtt{ARX}}(\Delta\gamma, \Delta\lambda \xrightarrow{r} \Delta\eta)$ (36) matches exactly the probability obtained experimentally $P_{\mathtt{exper}}$. The latter is computed over all 2^8 possible inputs. The values of the differences are in binary format.

#	$\Delta\gamma$	$\Delta\lambda$	$\Delta\eta$	r	$P_{\mathtt{exper}}$	$P_{\mathtt{ARX}}$	$P_{\mathtt{rotxor}}$
1	1000	0000	0001	1	0.500	0.500	0.344
2	1000	0000	0010	2	0.500	0.500	0.375
3	1000	0000	0110	2	0	0	0.125
4	1000	0000	1010	2	0	0	0.125
5	1000	0000	1110	2	0.500	0.500	0.375
6	1000	0000	1111	1	0.500	0.500	0.344
7	1000	0001	0000	1	0.500	0.500	0.344
8	1000	0010	0000	2	0.500	0.500	0.375
9	1000	0010	1000	2	0	0	0.125
10	1000	0110	0000	2	0	0	0.125
11	1000	0110	1000	2	0.500	0.500	0.375
12	1000	0111	1000	1	0.500	0.500	0.344
13	1000	1000	0010	2	0	0	0.125
14	1000	1000	0110	2	0.500	0.500	0.375
15	1000	1000	0111	1	0.500	0.500	0.344
16	1000	1000	1001	1	0.500	0.500	0.344
17	1000	1000	1010	2	0.500	0.500	0.375
18	1000	1000	1110	2	0	0	0.125
19	1000	1001	1000	1	0.500	0.500	0.344
20	1000	1010	0000	2	0	0	0.125
21	1000	1010	1000	2	0.500	0.500	0.375
22	1000	1110	0000	2	0.500	0.500	0.375
23	1000	1110	1000	2	0	0	0.125
24	1000	1111	0000	1	0.500	0.500	0.344

- $\mathrm{adp}^{\mathtt{ARX}}$: the probability computed using the proposed method (36)
- $P_{\mathtt{rotxor}}$: the probability computed as a product of the probabilities $\mathrm{adp}^{\lll}$ and $\mathrm{adp}^{\oplus}$ (20)

In the set of experiments on 4-bit words, we exhaustively searched over all possible combinations of input and output differences $\Delta\gamma, \Delta\lambda, \Delta\eta$ and over all non-zero rotation constants $r \in \{1, 2, 3\}$. We performed $12,288$ experiments in total. For each of them we computed $P_{\mathtt{exper}}$, $\mathrm{adp}^{\mathtt{ARX}}$ and $P_{\mathtt{rotxor}}$. The probability $P_{\mathtt{exper}}$ was computed over all 2^8 possible input words. In each experiment, the probability $\mathrm{adp}^{\mathtt{ARX}}$ was equal to $P_{\mathtt{exper}}$, while $P_{\mathtt{rotxor}}$ often deviated. The 24 cases in which the absolute deviation is higher than 0.1 are shown in Table 3.

We experimented over random 32-bit input and output differences of relatively low weight (less than 16). The probability $P_{\mathtt{exper}}$ was computed over 2^{22} random inputs. We performed 2^{10} experiments in total. For all of them, the estimation

Table 4. Comparing three ways of computing the additive differential probability ARX for 32-bit words. Shown are 11 selected cases for which the deviation of $P_{\texttt{rotxor}}$ (20) from the experimentally obtained value $P_{\texttt{exper}}$ (19) is high. The probability $\mathrm{adp}^{\texttt{ARX}}(\Delta\gamma, \Delta\lambda \xrightarrow{r} \Delta\eta)$ (36) closely follows the experimentally obtained value $P_{\texttt{exper}}$. The latter is computed over 2^{22} random inputs. Base-2 logarithms of the probabilities are given.

#	$\Delta\gamma$	$\Delta\lambda$	$\Delta\eta$	r	$P_{\texttt{exper}}$	$P_{\texttt{ARX}}$	$P_{\texttt{rotxor}}$
0	0x80000100	0x00000000	0x0007fc00	11	−2.58331	−2.58496	−4.17006
1	0x40000008	0x00000000	0x000001d0	6	−4.58591	−4.58496	−5.59061
2	0x80000008	0x04000000	0xfc000f00	9	−4.16817	−4.16711	−5.70768
3	0x40010001	0x04000000	0xd3ffc000	30	−5.90603	−5.91254	−6.60771
4	0xa2005800	0x00400000	0xf4000b00	29	−7.53935	−7.54954	−8.57279
5	0x45003700	0x00000000	0xc8ffbb00	16	−8.77902	−8.76145	−9.37302
6	0x4007800d	0x03800300	0x01e803f0	21	−11.14047	−11.17440	−11.86110
7	0xbf006400	0x00900050	0xf37ff9f0	28	−11.85917	−11.82987	−12.81410
8	0x8d00ec00	0x00a000f0	0xfbf7f870	27	−12.04435	−12.05139	−13.47130
9	0x7c005e00	0x00700080	0xffb3fe78	9	−9.96830	−9.98809	−11.36139
10	0xda008200	0x001000d0	0xe01d9f38	20	−15.05749	−15.10578	−15.77518
11	0xe4006600	0x00f00040	0xf0cff9e0	28	−15.04580	−15.04912	−15.32160

of the probability $\mathrm{adp}^{\texttt{ARX}}$ was closer to the experimentally obtained value $P_{\texttt{exper}}$ than to $P_{\texttt{rotxor}}$. A selection of 11 cases for which the absolute deviation from $P_{\texttt{rotxor}}$ was observed to be relatively high is shown in Table 4.

8 Conclusions

In this paper, we analyzed the probability $\mathrm{adp}^{\texttt{ARX}}$ with which additive differences propagate through the sequence of operations: modular addition, bit rotation and XOR. We proposed a method for the computation of $\mathrm{adp}^{\texttt{ARX}}$, based on the recently proposed concept of S-functions. The time complexity of our algorithm is linear in the word size n. To the best of our knowledge, our algorithm is the first to calculate $\mathrm{adp}^{\texttt{ARX}}$ efficiently for large n.

In Sect. 7, we observed that the estimated probability obtained by analyzing the components of ARX separately, can differ significantly from the actual probability. In our method, we analyze the three operations as a single operation (ARX). In this way, we obtain the exact probability $\mathrm{adp}^{\texttt{ARX}}$. Our algorithm can be used to evaluate the probability of differential characteristics for cryptographic algorithms more accurately.

An interesting topic for future research, is therefore to use our technique in the search of differential characteristics. Possible targets include several hash functions from NIST's ongoing SHA-3 competition, as well as stream ciphers (e.g. Salsa20), or block ciphers such as XTEA.

Acknowledgments. The authors would like to thank Vincent Rijmen for reviewing the draft paper and for providing important suggestions and comments. We also thank our colleagues at COSIC for the useful discussions, as well as the anonymous reviewers for their detailed comments.

References

1. Aumasson, J.-P., Henzen, L., Meier, W., Phan, R.C.-W.: SHA-3 proposal BLAKE. Submission to the NIST SHA-3 Competition, Round 2 (2008)
2. Bernstein, D.J.: The Salsa20 Family of Stream Ciphers. In: Robshaw, M.J.B., Billet, O. (eds.) [14], pp. 84–97
3. Bernstein, D.J.: CubeHash specification (2.B.1). Submission to the NIST SHA-3 Competition, Round 2 (2009)
4. Bresson, E., Canteaut, A., Chevallier-Mames, B., Clavier, C., Fuhr, T., Gouget, A., Icart, T., Misarsky, J.-F., Naya-Plasencia, M., Paillier, P., Pornin, T., Reinhard, J.-R., Thuillet, C., Videau, M.: Shabal, a Submission to NIST's Cryptographic Hash Algorithm Competition. Submission to the NIST SHA-3 Competition, Round 2 (2008)
5. Daum, M.: Cryptanalysis of Hash Functions of the MD4-Family. PhD thesis, Ruhr-Universität Bochum (2005)
6. Ferguson, N., Lucks, S., Schneier, B., Whiting, D., Bellare, M., Kohno, T., Callas, J., Walker, J.: The Skein Hash Function Family. Submission to the NIST SHA-3 Competition, Round 2 (2009)
7. Gligoroski, D., Klima, V., Knapskog, S.J., El-Hadedy, M., Amundsen, J., Mjølsnes, S.F.: Cryptographic Hash Function BLUE MIDNIGHT WISH. Submission to the NIST SHA-3 Competition, Round 2 (2009)
8. Leurent, G., Bouillaguet, C., Fouque, P.-A.: SIMD Is a Message Digest. Submission to the NIST SHA-3 Competition, Round 2 (2009)
9. Lipmaa, H., Moriai, S.: Efficient Algorithms for Computing Differential Properties of Addition. In: Matsui, M. (ed.) FSE 2001. LNCS, vol. 2355, pp. 336–350. Springer, Heidelberg (2002)
10. Lipmaa, H., Wallén, J., Dumas, P.: On the Additive Differential Probability of Exclusive-Or. In: Roy, B.K., Meier, W. (eds.) FSE 2004. LNCS, vol. 3017, pp. 317–331. Springer, Heidelberg (2004)
11. Mouha, N., Velichkov, V., De Cannière, C., Preneel, B.: The Differential Analysis of S-Functions. In: Biryukov, A., Gong, G., Stinson, D.R. (eds.) SAC 2010. LNCS, vol. 6544, pp. 36–56. Springer, Heidelberg (2011)
12. National Institute of Standards and Technology. Announcing Request for Candidate Algorithm Nominations for a New Cryptographic Hash Algorithm (SHA-3) Family. Federal Register 27(212), 62212–62220 (2007), `http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Nov07.pdf` (October 17, 2008)
13. Needham, R.M., Wheeler, D.J.: Tea extensions. Computer Laboratory, Cambridge University, England (1997), `http://www.movable-type.co.uk/scripts/xtea.pdf`
14. Robshaw, M.J.B., Billet, O. (eds.): New Stream Cipher Designs - The eSTREAM Finalists. LNCS, vol. 4986. Springer, Heidelberg (2008)
15. Staffelbach, O., Meier, W.: Cryptographic Significance of the Carry for Ciphers Based on Integer Addition. In: Menezes, A., Vanstone, S.A. (eds.) CRYPTO 1990. LNCS, vol. 537, pp. 601–613. Springer, Heidelberg (1991)
16. Wu, H.: The Stream Cipher HC-128. In: Robshaw, M.J.B., Billet, O. (eds.) [14], pp. 39–47

A Appendix

$$A_{000} = \begin{bmatrix} 0&1&1&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0 \\ 0&1&1&0&4&0&0&1 \\ 0&1&0&0&0&0&0&1 \\ 0&0&1&0&0&0&0&1 \\ 0&0&0&0&0&0&0&1 \end{bmatrix}, A_{001} = \begin{bmatrix} 4&0&0&1&0&1&1&0 \\ 0&0&0&1&0&1&0&0 \\ 0&0&0&1&0&0&1&0 \\ 0&0&0&1&0&0&0&0 \\ 0&0&0&0&0&1&1&0 \\ 0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&0 \end{bmatrix}, A_{010} = \begin{bmatrix} 1&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0 \\ 1&0&0&1&0&0&0&0 \\ 0&0&0&1&0&0&0&0 \\ 1&0&0&0&0&1&0&0 \\ 0&0&0&0&0&1&0&0 \\ 1&0&0&1&0&1&4&0 \\ 0&0&0&1&0&1&0&0 \end{bmatrix},$$

$$A_{011} = \begin{bmatrix} 0&1&0&0&1&0&0&0 \\ 0&1&0&0&0&0&0&0 \\ 0&1&4&0&1&0&0&1 \\ 0&1&0&0&0&0&0&1 \\ 0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&0&0&0 \\ 0&0&0&0&1&0&0&1 \\ 0&0&0&0&0&0&0&1 \end{bmatrix}, A_{100} = \begin{bmatrix} 1&0&0&0&0&0&0&0 \\ 1&0&0&1&0&0&0&0 \\ 0&0&0&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0 \\ 1&0&0&0&0&0&1&0 \\ 1&0&0&1&0&4&1&0 \\ 0&0&0&0&0&0&1&0 \\ 0&0&0&1&0&0&1&0 \end{bmatrix}, A_{101} = \begin{bmatrix} 0&0&1&0&1&0&0&0 \\ 0&4&1&0&1&0&0&1 \\ 0&0&1&0&0&0&0&0 \\ 0&0&1&0&0&0&0&1 \\ 0&0&0&0&1&0&0&0 \\ 0&0&0&0&1&0&0&1 \\ 0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&1 \end{bmatrix},$$

$$A_{110} = \begin{bmatrix} 0&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0 \\ 0&1&1&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0 \\ 0&1&0&0&1&0&0&0 \\ 0&0&1&0&1&0&0&0 \\ 0&1&1&0&1&0&0&4 \end{bmatrix}, A_{111} = \begin{bmatrix} 1&0&0&0&0&0&0&0 \\ 1&0&0&0&0&1&0&0 \\ 1&0&0&0&0&0&1&0 \\ 1&0&0&4&0&1&1&0 \\ 0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&1&1&0 \end{bmatrix}, \quad R = \begin{bmatrix} 0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0 \\ 1&0&1&0&1&0&1&0 \\ 0&1&0&1&0&1&0&1 \\ 0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0 \end{bmatrix}.$$

B Proof of Correctness of the Computation of $\mathbf{adp}^{\mathtt{ARX}}$

In this section, we provide the full proof of Theorem 1. We use the graph representation of an S-function [11]. For input words of size n, an S-function can be represented as a directed acyclic graph, composed of n bipartite subgraphs. Each bipartite subgraph corresponds to one of the eight adjacency matrices $A_q,\ q \in \{0, 1, \ldots, 7\}$. The vertices of the i-th subgraph are composed of the two disjoint sets of eight input states $S[i] \in \{0, 1, \ldots, 7\}$ and eight output states $S[i + 1] \in \{0, 1, \ldots, 7\}$. Furthermore, the output states of the i-th subgraph are input states for the $(i + 1)$-th subgraph. An edge between a vertex in $S[i]$ and a vertex in $S[i + 1]$ corresponds to a value of the tuple $(c_1[i], d_1[i + r])$ that results in the fixed output difference $\Delta e[i + r] = \Delta\eta[i + r]$. With this representation in mind, we state the following two lemmas before we proceed to the main theorem.

Lemma 1. *Let input differences $\Delta\gamma[i], \Delta\lambda[i+r]$ be given. Then, for every input value $(c_1[i], d_1[i + r])$ and input state $S[i]$, the output value $\Delta e[i + r]$ and the output state $S[i + 1]$ are uniquely determined.*

Proof. The proof follows directly from (23)-(25). □

Lemma 2. *The i-th subgraph in the graph representation of the* $\mathrm{adp}^{\mathtt{ARX}}$ *S-function (29) contains an edge if and only if* $\Delta e[i+r] = \Delta\eta[i+r]$

Proof. The statement holds by construction of the subgraphs. □

Theorem 1

$$2^{-2n} \sum_{j \in \{0,2,4,6\}} (L_j A_{w[n-1]} \cdots A_{w[n-r]} R A_{w[n-r-1]} \cdots A_{w[1]} A_{w[0]} C_j) = \frac{|\{(c_1, d_1) : \Delta e = \Delta\eta\}|}{|\{(c_1, d_1)\}|} \ . \tag{38}$$

Proof. Proving the statement of the theorem is equivalent to proving that the result computed by formula (36) is equal to the definition of $\mathrm{adp}^{\mathtt{ARX}}$ (19). Consider the S-function for $\mathrm{adp}^{\mathtt{ARX}}$ (29) and the i-th subgraph of its graph representation. Fix the inputs $\Delta\gamma[i], \Delta\lambda[i+r]$. From Lemma 1, it follows that every edge in the subgraph corresponds to a distinct pair of inputs $(c_1[i], d_1[i+r]), (c_2[i], d_2[i+r])$ that satisfies the input differences $(\Delta\gamma[i], \Delta\lambda[i+r])$. From Lemma 2, it follows that the subgraph contains only those among all edges, for which the pair of inputs satisfies also the output difference $\Delta\eta[i+r]$. Consider next the graph composed of all n subgraphs. A path in this graph is composed of n edges: one edge from each subgraph. For bit position i, one edge corresponds to distinct pairs $(c_1[i], d_1[i+r]), (c_2[i], d_2[i+r])$ that satisfy differences $\Delta\gamma[i], \Delta\lambda[i+r], \Delta\eta[i+r]$. Therefore, a path composed of n edges will correspond to distinct pairs $(c_1, d_1), (c_2, d_2)$ that satisfy the n-bit differences $\Delta\gamma, \Delta\lambda, \Delta\eta$. It follows that the number of paths in the S-function graph is equal to the number of pairs of inputs that satisfy both the input and the output differences. The number of paths that connect input state $S[0] = u \in \{0, \ldots, 7\}$ to output state $S[n-1] = v \in \{0, \ldots, 7\}$ is equal to the value of the element in column u and row v of the matrix A, denoted by $A_{u,v}$ with indexing starting from zero. The matrix A is obtained by multiplying the n adjacency matrices corresponding to each of the n subgraphs

$$A = A_{w[n-1]} \cdots A_{w[n-r]} R A_{w[n-r-1]} \cdots A_{w[1]} A_{w[0]} \ , \tag{39}$$

where R is the projection matrix derived in Sect. 5. In Sect. 5, it was shown that due to the bit rotation in the ARX operation, the only valid initial states for the S-function are $S[0] = u \in \{0, 2, 4, 6\}$. Their corresponding valid final states are $S[n-1] = u$ and $S[n-1] = u + 1$. Therefore the number of paths connecting valid input and output states is equal to the sum of elements $A_{u,v}$ $u \in \{0, 2, 4, 6\}, v \in \{u, u+1\}$ of A:

$$\sum_{u \in \{0,2,4,6\}} \sum_{v \in \{u, u+1\}} A_{u,v} = \sum_{j \in \{0,2,4,6\}} L_j A C_j \ , \tag{40}$$

where C_j and L_j are the same as in (36). It remains to prove that (40) is equal to $|\{(c_1, d_1) : \Delta e = \Delta\eta\}|$. For this it is enough to show that none of the paths corresponding to $A_{u,v}$ overlap. This is indeed the case since the four initial states u do not overlap (no two values of u are equal) and each of them ends in a set of final states so that no two sets $\{u, u+1\}$ overlap. From this, and because $|\{(c_1, d_1)\}| = 2^{2n}$, it follows that

$$2^{-2n} \sum_{j \in \{0,2,4,6\}} L_j A C_j = \frac{|\{(c_1, d_1) : \Delta e = \Delta\eta\}|}{|\{(c_1, d_1)\}|} = \mathrm{adp}^{\mathtt{ARX}}(\Delta\gamma, \Delta\lambda \xrightarrow{r} \Delta\eta).$$

$\square$