UNAF: A Special Set of Additive Differences with Application to the Differential Analysis of ARX[⋆]

Vesselin Velichkov[1,2,3,⋆⋆], Nicky Mouha[1,3,⋆⋆⋆], Christophe De Cannière[1,3,†], and Bart Preneel[1,3]

[1] Department of Electrical Engineering ESAT/SCD-COSIC, Katholieke Universiteit Leuven. Kasteelpark Arenberg 10, B-3001 Heverlee, Belgium.
[2] Laboratory of Algorithmics, Cryptology and Security (LACS), University of Luxembourg, Luxembourg.
[3] Interdisciplinary Institute for BroadBand Technology (IBBT), Belgium.
{Vesselin.Velichkov,Nicky.Mouha}@esat.kuleuven.be

Abstract. Due to their fast performance in software, an increasing number of cryptographic primitives are constructed using the operations addition modulo 2^n, bit rotation and XOR (ARX). However, the resistance of ARX-based ciphers against differential cryptanalysis is not well understood. In this paper, we propose a new tool for evaluating more accurately the probabilities of additive differentials over multiple rounds of a cryptographic primitive. First, we introduce a special set of additive differences, called UNAF (unsigned non-adjacent form) differences. Then, we show how to apply them to find good differential trails using an algorithm for the automatic search for differentials. Finally, we describe a key-recovery attack on stream cipher Salsa20 reduced to five rounds, based on UNAF differences.

Keywords: UNAF, ARX, Salsa20, additive differential probability, differential cryptanalysis

1 Introduction

Differential cryptanalysis [4] and linear cryptanalysis [14] have shown to be two of the most powerful techniques in the cryptanalysis of symmetric-key cryptographic primitives. Security against linear and differential cryptanalysis is therefore typically a major design criterion for modern ciphers. An example of this is the wide-trail design strategy, used to provide provable resistance against linear and differential cryptanalysis for the AES block cipher [6].

In order to achieve a fast performance in software, an increasing number of cryptographic primitives are built using the operations addition modulo 2^n, rotation and XOR (ARX). Examples include the block cipher FEAL [17], the Salsa20 stream cipher family [3], as well as the SHA-3 finalists BLAKE [2] and Skein [9]. Although ARX-based algorithms are very popular, their resistance to differential cryptanalysis [4] is not well understood.

The probability with which differences propagate through a sequence of operations must be calculated efficiently and accurately, in order to correctly assess the security of a cipher against differential cryptanalysis. Lipmaa et al. studied the xor-differential probability of addition (xdp^+) in [12], and the additive differential probability of XOR ($\mathrm{adp}^{\oplus}$) in [13]. These results were generalized using the S-functions framework, introduced by Mouha et al. [15].

As shown by Velichkov et al. [18], the additive differential probability of ARX ($\mathrm{adp}^{\mathtt{ARX}}$) can differ significantly from the multiplication of the differential probability of the separate components – addition, rotation and XOR. Although an algorithm was proposed in [18] for the exact

[⋆] 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).

Table 1. Notation.

Symbol	**Meaning**
n	Number of bits in a 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)
$\|x\|$	The absolute value of x
$\overline{x}$	The negation of x i.e. $\overline{x} = -x$ (e.g. $\overline{1} = -1$)
$\#A$	Number of elements in the set A
$+$, $-$	Addition modulo 2^n, subtraction modulo 2^n
$\oplus$	Exclusive-OR (XOR)
$\lll t$	Left bit rotation by t positions
$\alpha \to \beta$	Input difference α propagates to output difference β
w_i^r	32-bit word i from the input state to round $r+1$ of Salsa20
Δ_i^r	Additive difference in word i of the input to round $r+1$ of Salsa20
0_i^r	Zero difference in word i of the input to round $r+1$ of Salsa20
$\{\Delta^U\}_i^r$	UNAF difference in word i of the input to round $r+1$ of Salsa20
ARX	The sequence of the operations: $+, \lll, \oplus$ as a single operation
HW(x)	Hamming weight of x (number of non-zero bits in x)

calculation of $\text{adp}^{\texttt{ARX}}$, unfortunately their method does not scale to analyze larger components. The accurate calculation of the probability of a differential characteristic therefore still remains an open problem for ARX constructions.

In this paper we take a different approach. Namely, we do not calculate the *exact* differential probability of a component consisting of more than one ARX operations. Instead, we multiply the differential probabilities of several ARX operations in order to estimate the total probability. As we want to avoid that this calculation differs significantly from the actual probability (e.g. due to dependencies between the inputs as noted in [18]), we propose to use a new type of difference: the UNAF difference, which represents a set of specially chosen additive differences.

We apply UNAF differences to the cryptanalysis of the ARX-based stream cipher Salsa20. A general algorithm for automatic search of differentials is briefly discussed. We apply it to find several differentials for three rounds of Salsa20. By multiplying the probabilities $\text{adp}^{\texttt{ARX}}$ of separate ARX components, we estimate that the best differential has a probability of 2^{-10}. Using UNAF differences, the same probability is evaluated as 2^{-4}. Experimentally, we estimate the probability of this differential to be $2^{-3.39}$. We observe that the probability obtained using UNAF differences is much closer to the experimental value.

Finally, we apply UNAF differences to mount key-recovery attack on a version of Salsa20 reduced to 5 rounds. Note that this is not the best known attack on Salsa20. It is therefore provided only as a demonstration of a practical application of UNAF differences. Furthermore, we expect that our attack can be extended to more rounds.

The outline of the paper is as follows. In Sect. 2, we describe the UNAF framework. It is applied to the differential analysis of stream cipher Salsa20 in Sect. 3. Sect. 4 concludes the paper. Notation is defined in Table 1.

2 The UNAF Framework

In this section, we describe the UNAF framework. We define UNAF differences and state the main UNAF theorem. The UNAF differential probability of ARX ($\text{udp}^{\texttt{ARX}}$) is defined and a general algorithm for the automatic search for high-probability differentials is briefly discussed.

2.1 Preliminaries

Before we give the formal definition of UNAF differences, we first recall a few related concepts: the binary-signed digit (BSD) difference and the non-adjacent form (NAF) difference.

Definition 1. (BSD difference) *A BSD difference is a difference whose bits are signed and take values in the set* $\{\bar{1}, 0, 1\}$*:*

$$\Delta^{\pm}a : \Delta^{\pm}a[i] = (a_2[i] - a_1[i]) \in \{\bar{1}, 0, 1\}, \quad 0 \le i < n \ . \tag{1}$$

An additive difference $\Delta^{+}a$ can be composed of more than one BSD difference $\Delta^{\pm}a$. From any BSD difference, the corresponding additive difference can be computed as: $\Delta^{+}a = \sum_{i=0}^{n-1} \Delta^{\pm}a[i] \cdot 2^i$.

All BSD differences corresponding to $\Delta^{+}a$ can be obtained by replacing 01 with $1\bar{1}$ and vice versa and by replacing $0\bar{1}$ with $\bar{1}1$ and vice versa [7, 16]. Note also that the number of pairs (a_1, a_2) that satisfy the n-bit difference $\Delta^{+}a$ is 2^n, while the number of pairs that satisfy any of its BSD differences $\Delta^{\pm}a$ is 2^k, where k is the number of zeros in the word $\Delta^{\pm}a$. Therefore, the following inequality holds: $2^k \le 2^n$, $k = n - \mathrm{HW}(\Delta^{\pm}a)$.

The non-adjacent form (NAF) difference is a special BSD difference and is defined as follows:

Definition 2. (NAF) *A NAF (non-adjacent form) difference is a BSD difference in which no two adjacent bits are non-zero:*

$$\Delta^{N}a : \quad \nexists i : \quad (\Delta^{N}a[i] \ne 0) \wedge (\Delta^{N}a[i+1] \ne 0), \quad 0 \le i < n-1 \ . \tag{2}$$

For every additive difference $\Delta^{+}a$, there is exactly one NAF difference $\Delta^{N}a$ (ignoring the sign of the MSB). No other BSD difference has a lower Hamming weight than $\Delta^{N}a$ [16]. We illustrate this with the following example:

Example 1. Let $n = 4$ and $\Delta^{+}a = 3$. Then all possible BSD differences corresponding to $\Delta^{+}a$ are 0011, $010\bar{1}$, $01\bar{1}1$, $1\bar{1}\bar{1}1$, $\bar{1}\bar{1}\bar{1}1$, $1\bar{1}0\bar{1}$ and $\bar{1}\bar{1}0\bar{1}$. Of them, only $010\bar{1}$ is in non-adjacent form (NAF). It also has the lowest Hamming weight among all BSD differences, namely 2.

By enumerating all possible combinations of signs of the non-zero bits of $\Delta^{N}a$, we can construct a special set of additive differences. What is special about this set, is that all of its elements correspond to the same unsigned NAF difference. This set is a UNAF difference and is denoted by $\Delta^{U}a$. More formally:

Definition 3. (UNAF) *A UNAF difference is a set of additive differences that correspond to the same unsigned NAF difference (i.e. a NAF difference with the signs ignored):*

$$\Delta^{U}a = \{\Delta^{+}x : |\Delta^{N}x| = |\Delta^{N}a|\} \ . \tag{3}$$

It is easy to see that the size of the UNAF set $\Delta^{U}a$ is 2^k, where k is the Hamming weight of the n-bit word $\Delta^{N}a$, excluding the MSB. We further clarify the concept of a UNAF difference with the following example:

Example 2. Consider again an example where $n = 4$. Let $\Delta^{+}a = 3$, thus $\Delta^{N}a = 010\bar{1}$. Then, $\Delta^{U}a = \{\Delta^{+}x_1 = 3, \Delta^{+}x_2 = -3, \Delta^{+}x_3 = 5, \Delta^{+}x_4 = -5\}$. This follows from $|\Delta^{N}x_1| = |\Delta^{N}x_2| = |\Delta^{N}x_3| = |\Delta^{N}x_4| = |\Delta^{N}a|$, because $|010\bar{1}| = |0\bar{1}01| = |0101| = |0\bar{1}0\bar{1}| = 0101$.

2.2 Main UNAF Theorem

The main UNAF theorem provides the motivation for applying UNAF differences to the differential analysis of ARX. Before we state it, we define the additive differential probability of `XOR` ($\mathrm{adp}^{\oplus}$).

The differential probability of the operation `XOR`, when differences are expressed using addition modulo 2^n, is denoted by $\mathrm{adp}^{\oplus}$. For fixed additive differences α, β and γ, $\mathrm{adp}^{\oplus}$ is equal to the number of pairs (a_1, b_1) for which the equality $((a_1 + \alpha) \oplus (b_1 + \beta)) - (a_1 \oplus b_1) = \gamma$ holds, divided by the total number of such pairs. More formally, $\mathrm{adp}^{\oplus}(\alpha, \beta \to \gamma)$ is defined as:

Definition 4. ($\mathrm{adp}^{\oplus}$)

$$\begin{aligned}\mathrm{adp}^{\oplus}(\alpha, \beta \to \gamma) &= \frac{\#\{(a_1, b_1) : c_2 - c_1 = \gamma\}}{\#\{(a_1, b_1)\}} \\ &= 2^{-2n} \cdot \#\{(a_1, b_1) : c_2 - c_1 = \gamma\} \ , \end{aligned} \tag{4}$$

where $c_1 = a_1 \oplus b_1$, $c_2 = (a_1 + \alpha) \oplus (b_1 + \beta)$ *and* 2^{2n} *is the total number of pairs* (a_1, b_1).

Efficient algorithms for the computation of $\mathrm{adp}^{\oplus}$ were studied in [13, 15]. Next we state the main UNAF theorem. Its proof is given in Appendix A.

Theorem 1. (Main UNAF theorem) *If the probability with which input additive differences Δ^+a and Δ^+b propagate to output difference Δ^+c through `XOR` is non-zero, then the probability with which any of the input additive differences belonging to the corresponding UNAF sets resp. $\Delta^U a$ and $\Delta^U b$ propagate to any of the output additive differences belonging to the UNAF set $\Delta^U c$ is also non-zero:*

$$\begin{aligned}\mathrm{adp}^{\oplus}(\Delta^+a, \Delta^+b \to \Delta^+c) > 0 &\implies \mathrm{adp}^{\oplus}(\Delta^+a_i, \Delta^+b_j \to \Delta^+c_k) > 0 \ , \\ &\forall i, j, k : \Delta^+a_i \in \Delta^U a, \Delta^+b_j \in \Delta^U b, \Delta^+c_k \in \Delta^U c \ . \end{aligned} \tag{5}$$

Theorem 1 states that if a given additive differential is possible w.r.t. the `XOR` operation, then all additive differentials whose inputs and outputs belong to the same UNAF sets, are also possible. This is illustrated with the following example.

Example 3. Let $n = 4$ and $\Delta^+a = 5$, $\Delta^+b = 1$, $\Delta^+c = 6$. Because $\mathrm{adp}^{\oplus}(5, 1 \to 6) = 0.15625 > 0$, we can use Theorem 1 to show that $\mathrm{adp}^{\oplus}(\Delta^+a_i, \Delta^+b_j \to \Delta^+c_k) > 0$ for any $\Delta^+a_i \in \Delta^U a = \{3, -3, 5, -5\}$, $\Delta^+b_j \in \Delta^U b = \{1, -1\}$ and $\Delta^+c_k \in \Delta^U c = \{6, -6\}$.

In the next section we investigate the probability with which UNAF differences propagate through the `ARX` operation.

2.3 The UNAF Differential Probability of `ARX`

The UNAF differential probability of `ARX` represents the probability with which the sets of input additive differences $\Delta^U a$, $\Delta^U b$ and $\Delta^U d$ propagate to the set of output additive differences $\Delta^U e$. It is defined as:

Definition 5. ($\mathrm{udp}^{\mathtt{ARX}}$)

$$\begin{aligned}&\mathrm{udp}^{\mathtt{ARX}}(\Delta^U a, \Delta^U b, \Delta^U d \xrightarrow{t} \Delta^U e) = \\ &\frac{\#\{(a_1, b_1, d_1) : \Delta^+a \in \Delta^U a, \Delta^+b \in \Delta^U b, \Delta^+d \in \Delta^U d, \Delta^+e \in \Delta^U e\}}{\#\{(a_1, b_1, d_1) : \Delta^+a \in \Delta^U a, \Delta^+b \in \Delta^U b, \Delta^+d \in \Delta^U d\}} \ , \end{aligned} \tag{6}$$

where

$$\Delta^+e = e_2 - e_1 = \mathtt{ARX}(a_1 + \Delta^+a, b_1 + \Delta^+b, d_1 + \Delta^+d, t) - \mathtt{ARX}(a_1, b_1, d_1, t),$$

and $\mathtt{ARX}(x, y, z, t) = ((x + y) \lll t) \oplus z$.

The probability $\mathrm{udp}^{\mathtt{ARX}}$ is computed using a method conceptually similar to the one proposed for the computation of $\mathrm{adp}^{\mathtt{ARX}}$ in [18]. The main difference is that in this case we are dealing with *sets* of input and output additive differences. Details on this computation are provided in Appendix B.

2.4 An Algorithm for Finding the Best Output Difference

To demonstrate how the UNAF framework can be used to construct high-probability differential characteristics, we have developed a general algorithm for the automatic search of differentials. It is capable of computing the highest probability output difference from a given operation. The proposed algorithm is applicable to any type of difference and any operation. The only condition is that the propagation of the difference through the operation can be represented as an S-function. The method to find the best output difference is based on the A* search algorithm [11].

Space constraints do not allow us to present the algorithm here in detail. However, a full description of the algorithm accompanied by pseudo-code can be found in Appendix C. Furthermore, a software toolkit that implements this algorithm is available.[4]

In the following sections we describe an application of the algorithm and of UNAF differences to the differential analysis of stream cipher Salsa20.

3 Applications

We describe several applications of the UNAF framework to the differential analysis of stream cipher Salsa20. UNAF differences can be used to obtain more accurate estimations of the probabilities of differentials through multiple rounds of ARX operations. We describe a key-recovery attack using UNAF differentials on a version of Salsa20, reduced to 5 rounds.

3.1 Description of Salsa20

Salsa20 is a stream cipher proposed by Bernstein in [3]. It is one of the finalists of the eSTREAM competition [8]. Salsa20 operates on 32-bit words. The inputs are a 256-bit key $(k_0, k_1, \ldots, k_7)$, a 64-bit nonce (v_0, v_1), a 64-bit counter (t_0, t_1) and four predefined 32-bit constants c_0, c_1, c_2, c_3. These inputs are mapped to a two-dimensional square matrix as follows:

$$\begin{bmatrix} c_0 & k_0 & k_1 & k_2 \\ k_3 & c_1 & v_0 & v_1 \\ t_0 & t_1 & c_2 & k_4 \\ k_5 & k_6 & k_7 & c_3 \end{bmatrix} \rightarrow \begin{bmatrix} w_0^0 & w_1^0 & w_2^0 & w_3^0 \\ w_4^0 & w_5^0 & w_6^0 & w_7^0 \\ w_8^0 & w_9^0 & w_{10}^0 & w_{11}^0 \\ w_{12}^0 & w_{13}^0 & w_{14}^0 & w_{15}^0 \end{bmatrix} . \tag{7}$$

The basic operation of Salsa20 is the *quarterround*. One *quarterround* transforms four of the input words to round $r+1$: $w_0^r, w_1^r, w_2^r, w_3^r$ into four output words: $w_0^{r+1}, w_1^{r+1}, w_2^{r+1}, w_3^{r+1}$ by the means of four consecutive ARX operations:

$$w_1^{r+1} = w_1^r \oplus ((w_0^r + w_3^r) \lll 7) = \mathtt{ARX}(w_0^r, w_3^r, w_1^r, 7) \ , \tag{8}$$
$$w_2^{r+1} = w_2^r \oplus ((w_1^{r+1} + w_0^r) \lll 9) = \mathtt{ARX}(w_1^{r+1}, w_0^r, w_2^r, 9) \ , \tag{9}$$
$$w_3^{r+1} = w_3^r \oplus ((w_2^{r+1} + w_1^{r+1}) \lll 13) = \mathtt{ARX}(w_2^{r+1}, w_1^{r+1}, w_3^r, 13) \ , \tag{10}$$
$$w_0^{r+1} = w_0^r \oplus ((w_3^{r+1} + w_2^{r+1}) \lll 18) = \mathtt{ARX}(w_3^{r+1}, w_2^{r+1}, w_0^r, 18) \ . \tag{11}$$

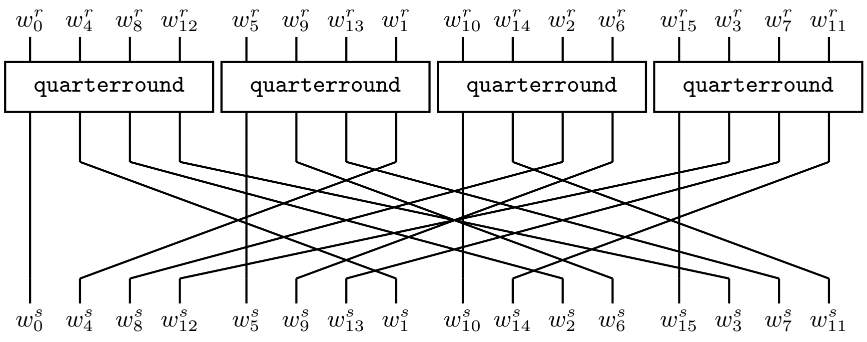

w_0^r w_4^r w_8^r w_{12}^r w_5^r w_9^r w_{13}^r w_1^r w_{10}^r w_{14}^r w_2^r w_6^r w_{15}^r w_3^r w_7^r w_{11}^r
quarterround
quarterround
quarterround
quarterround
w_0^s w_4^s w_8^s w_{12}^s w_5^s w_9^s w_{13}^s w_1^s w_{10}^s w_{14}^s w_2^s w_6^s w_{15}^s w_3^s w_7^s w_{11}^s

Fig. 1. Round $s = r + 1$ of Salsa20.

One *round* of Salsa20 consists of four parallel applications of the *quarterround* transformation. Each transformation is applied to the elements (in permuted order) of one of the four columns of the input state matrix, followed by a permutation of the words, as shown on Fig. 1.

Salsa20 has a total of 20 rounds, although versions with eight and twelve rounds have been proposed, resp. Salsa20/8 and Salsa20/12. The output state after the last round is added to the initial input state by means of a feed-forward operation. This produces sixteen 32-bit words (512 bits) of key stream.

3.2 Estimating the Probability of Differentials Using UNAF Differences

We apply the algorithm of Sect. 2.4 to search for high probability differential characteristics in Salsa20. We use a greedy strategy in which at every ARX operation we select the output UNAF difference with the highest probability, before proceeding with the next ARX operation. In this way we find the following truncated differential for three rounds:

$$\Delta_8^0 = \texttt{0x80000000} \rightarrow \Delta_9^3 = \texttt{0x80000000} \ . \tag{12}$$

The expression (12) implies that all words of the input state have zero difference, except for the word at position 8, which has difference 0x80000000. A three round differential characteristic that satisfies (12) is shown on Fig. 2. The probability with which the differential (12) holds, obtained experimentally over 2^{20} chosen plaintexts, is $p_{\text{exper}} = 2^{-3.39}$.

We compute two theoretical estimations of p_{exper}. The first estimation is based on single additive differences and is denoted $\hat{p}_{\text{add}}$. It is computed as a multiplication of $\text{adp}^{\texttt{ARX}}$ probabilities:

$$\hat{p}_{\text{add}} = \prod \text{adp}^{\texttt{ARX}} = 2^{-10} \ . \tag{13}$$

The second estimation of p_{exper} is based on UNAF differences and is denoted $\hat{p}_{\text{unaf}}$. It is computed as a multiplication of $\text{udp}^{\texttt{ARX}}$ probabilities:

$$\hat{p}_{\text{unaf}} = \prod \text{udp}^{\texttt{ARX}} = 2^{-4} \ . \tag{14}$$

The computations (13) and (14) are shown in Table 2 and Table 3 respectively.

Clearly $\hat{p}_{\text{unaf}}$ is a better estimation of p_{exper} than $\hat{p}_{\text{add}}$. The reason is that multiple differential characteristics connect the input and output differences of the differential (12). The estimation $\hat{p}_{\text{add}}$ is based upon a single one among all possible characteristics, while the estimation $\hat{p}_{\text{unaf}}$ takes into account several characteristics at once. This effect is illustrated in Fig. 3. Note that

[4] `http://www.ecrypt.eu.org/tools/s-function-toolkit`

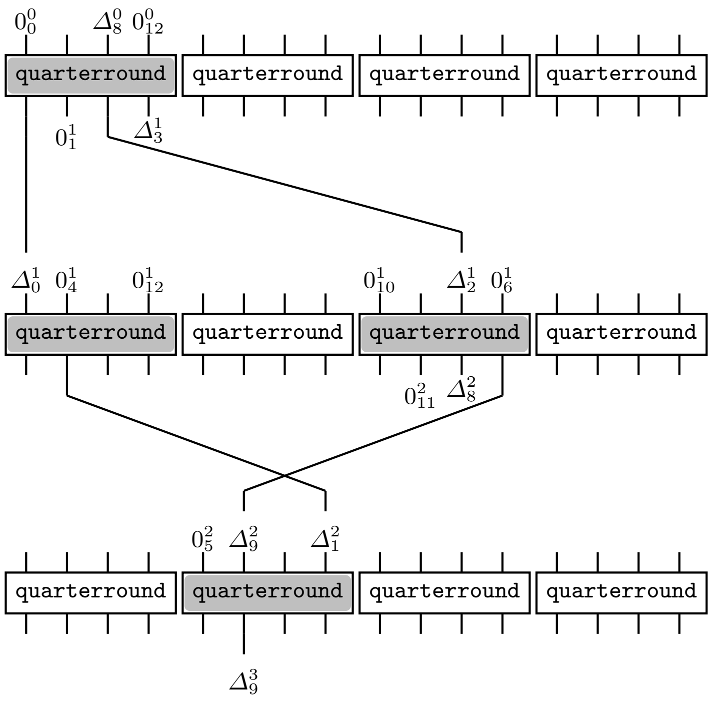


Fig. 2. Three round differential characteristic satisfying the differential $\Delta_8^0 \rightarrow \Delta_9^3$.

the input $\{\Delta^U\}_8^0$ and output $\{\Delta^U\}_9^3$ UNAF sets contain a single element – the additive difference 80000000. Because of that $\{\Delta^U\}_8^0 = \Delta_8^0$ and $\{\Delta^U\}_9^3 = \Delta_9^3$ and therefore the estimations (13) and (14) can be compared to each other.

In the case where the output UNAF set contains more than one element (i.e. $\{\Delta^U\}_9^3 \neq \Delta_9^3$), we propose to divide the resulting probability by the size of the output UNAF set $\#\Delta^U$:

$$\hat{p}_{\text{unaf}} = \frac{\prod \text{udp}^{\texttt{ARX}}}{\#\Delta^U} \quad . \tag{15}$$

The estimation (15) is based on the assumption that all additive differences from the output UNAF set Δ^U hold with approximately the same (or very close) probabilities. For the case of Salsa20, our experiments confirm this assumption.

We use (15) to estimate the probabilities with which several differences from the output state after Salsa20/3 hold, given input UNAF difference $\{\Delta^U\}_8^0 = \texttt{0x80000000}$. The results are shown in Table 4 and in Fig. 4.

The results presented in Table 4 and Fig. 4 show that although the probability estimations $\hat{p}_{\text{unaf}}/\#\Delta^U$ computed using UNAF differences with (15) deviate from the values obtained experimentally p_{exper}, they are still more accurate than the estimations $\hat{p}_{\text{add}}$ based on single additive differences and computed with (13).

3.3 Key-recovery Attack on Salsa20/5

In this section, we apply UNAF differences to mount a key-recovery attack on a version of stream cipher Salsa20 reduced to 5 rounds, denoted as Salsa20/5. Although its complexity is lower than exhaustive key search, the attack does not improve the best known attack on the cipher. Therefore it is described only as a demonstration of a practical application of UNAF differences.

Table 2. The estimated probability $\hat{p}_{\text{add}}$ (13) of the differential (12); $\text{adp}^{\texttt{ARX}}$ refers to $\text{adp}^{\texttt{ARX}}((\Delta^{+}a+\Delta^{+}b), \Delta^{+}d \xrightarrow{t} \Delta^{+}e)$.

Δ	$\Delta^{+}a$	$\Delta^{+}b$	$\Delta^{+}d$	t	$\Delta^{+}e = \Delta$	$\text{adp}^{\texttt{ARX}}$
Δ_2^1	0	0	80000000	9	80000000	1
Δ_3^1	80000000	0	0	13	fffff000	2^{-1}
Δ_0^1	fffff000	80000000	0	18	40020000	$2^{-2.41}$
Δ_1^2	40020000	0	0	7	01000020	$2^{-2.99}$
Δ_8^2	0	0	80000000	9	80000000	1
Δ_9^2	80000000	0	0	13	fffff000	2^{-1}
Δ_9^3	0	01000020	fffff000	7	80000000	$2^{-2.58}$
						$\hat{p}_{\text{add}} = 2^{-10}$

Table 3. The estimated probability $\hat{p}_{\text{unaf}}$ (14) of the differential (12); $\text{udp}^{\texttt{ARX}}$ refers to $\text{udp}^{\texttt{ARX}}(\Delta^{U}a, \Delta^{U}b, \Delta^{U}d \xrightarrow{t} \Delta^{U}e)$.

Δ^{U}	$\Delta^{U}a$	$\Delta^{U}b$	$\Delta^{U}d$	t	$\Delta^{U}e = \Delta^{U}$	$\text{udp}^{\texttt{ARX}}$
$\{\Delta^{U}\}_2^1$	0	0	80000000	9	80000000	1
$\{\Delta^{U}\}_3^1$	80000000	0	0	13	00001000	1
$\{\Delta^{U}\}_0^1$	00001000	80000000	0	18	40020000	$2^{-0.41}$
$\{\Delta^{U}\}_1^2$	40020000	0	0	7	01000020	$2^{-0.99}$
$\{\Delta^{U}\}_8^2$	0	0	80000000	9	80000000	1
$\{\Delta^{U}\}_9^2$	80000000	0	0	13	00001000	1
$\{\Delta^{U}\}_9^3$	0	01000020	00001000	7	80000000	$2^{-2.58}$
						$\hat{p}_{\text{unaf}} = 2^{-4}$

Using the best-first search algorithm from Sect. 2.4 we find the following UNAF differential for 3 rounds of Salsa20:

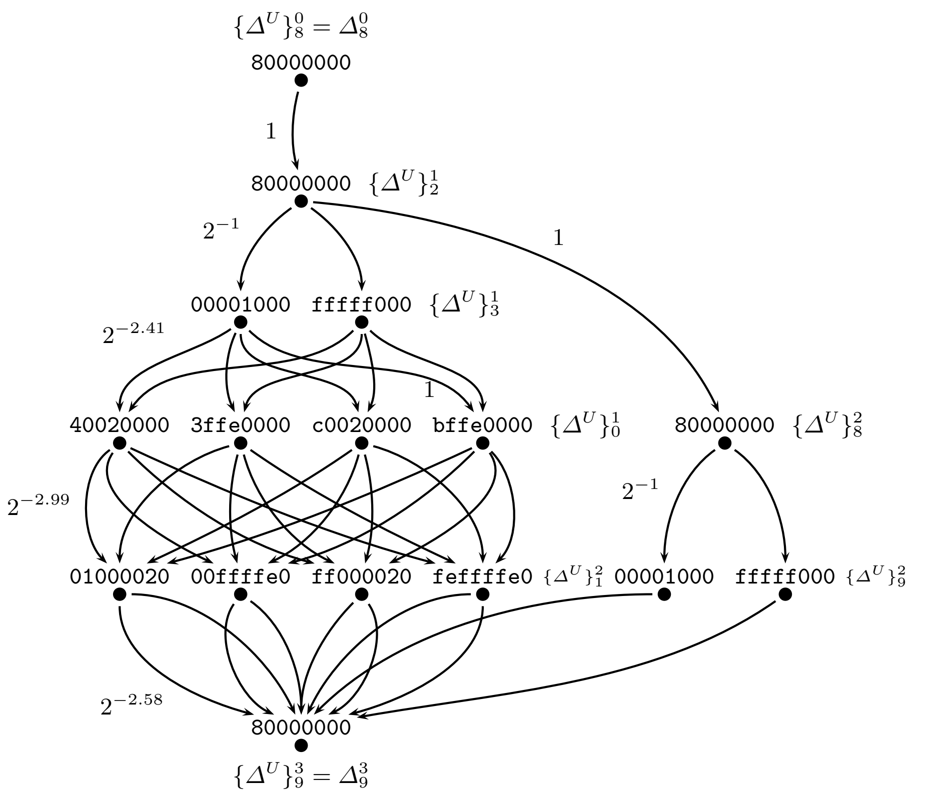
$$\{\Delta^{U}\}_8^0 = \texttt{0x80000000} \rightarrow \{\Delta^{U}\}_{11}^3 = \texttt{0x01000024} \ . \tag{16}$$

The input UNAF set $\{\Delta^{U}\}_8^0 = \texttt{0x80000000}$ consists of one element: the additive difference `0x80000000`. The output UNAF set $\{\Delta^{U}\}_{11}^3 = \texttt{0x01000024}$ contains the following 2^3 additive differences: `0x01000024`, `0x0100001c`, `0x00ffffe4`, `0x00ffffdc`, `0xff000024`, `0xff00001c`, `0xfeffffe4`, `0xfeffffdc`. The probability that an additive difference Δ_{11}^3 falls into the set $\{\Delta^{U}\}_{11}^3$ was determined experimentally to be $p_{\text{exper}} = 2^{-3.38}$.

In our attack, we first invert the feed-forward operation to compute the differences Δ_5^5, $\Delta_6^5, \ldots, \Delta_{10}^5$ of the state after round 5. Next, we guess 5 of the 8 words of the secret key, in order to compute the differences Δ_1^5,Δ_2^5,Δ_3^5,Δ_4^5,Δ_{11}^5. Therefore, we do not only know the differences Δ_1^5,Δ_2^5,…,Δ_{11}^5, but also the corresponding values of the word pairs. This allows us to compute the differences Δ_{12}^4,Δ_{13}^4,Δ_{14}^4 from the state after round 4. Using the latter, we can finally compute the UNAF difference $\{\Delta^{U}\}_{11}^3$. If it is equal to `0x01000024`, then our guess of the key words was correct with some probability. This process is illustrated in Appendix D.

Since the probability of the differential (16) is $2^{-3.38} \geq 2^{-4}$, from $M = 2^6$ chosen plaintext pairs we expect that $2^{-4} \cdot 2^6 = 2^2 = 4$ pairs will follow the differential (i.e. will satisfy the output difference $\{\Delta^{U}\}_{11}^3$).

We assume that a pair encrypted under a wrong key results in a uniformly random difference. The probability that this difference falls into the set $\{\Delta^{U}\}_{11}^3$ is $P_{\text{rand}} = 2^3/2^{32} = 2^{-29}$. Therefore the probability that at least 4 plaintext pairs turn out to be all false positives (i.e. they satisfy the differential, but are encrypted under a wrong key) can be calculated using the binomial



$\{\varDelta^U\}^0_8 = \varDelta^0_8$
80000000
1
80000000 $\{\varDelta^U\}^1_2$
2^{-1}
1
00001000 fffff000 $\{\varDelta^U\}^1_3$
$2^{-2.41}$
1
40020000 3ffe0000 c0020000 bffe0000 $\{\varDelta^U\}^1_0$
80000000 $\{\varDelta^U\}^2_8$
$2^{-2.99}$
2^{-1}
01000020 00ffffe0 ff000020 feffffe0 $\{\varDelta^U\}^2_1$
00001000 fffff000 $\{\varDelta^U\}^2_9$
$2^{-2.58}$
80000000
$\{\varDelta^U\}^3_9 = \varDelta^3_9$

Fig. 3. A single UNAF characteristic, satisfying the differential $\varDelta^0_8 \rightarrow \varDelta^3_9$. It is composed of multiple additive characteristics.

distribution:

$$\sum_{i=4}^{64} \binom{64}{i} (2^{-29})^i (1 - 2^{-29})^{64-i} \approx 2^{-96.72} \quad . \tag{17}$$

As explained, because we guess 160 bits (5 words) of the secret key, in the attack we have to make 2^{160} guesses. For each guess, we encrypt 2^6 chosen plaintext pairs and we partially decrypt the resulting ciphertext pairs for 2 rounds in order to compute the output difference. From 2^{160} guesses, the expected number of wrong keys that result in at least 4 pairs with the right difference is $2^{-96.72} \cdot 2^{160} \approx 2^{63}$. For each of those keys, we guess the remaining 96 bits (3 words) i.e. we make 2^{96} guesses per candidate key. For each guess we encrypt one plaintext pair (i.e. two encryptions are performed) under the full key and check if the encryption matches the corresponding ciphertext pair. This results in $2 \cdot 2^{63} \cdot 2^{96} = 2^{160}$ additional operations. Thus we estimate the total number of encryptions of our attack to be:

$$2 \cdot 2^6 \cdot 2^{160} + 2 \cdot 2^{63} \cdot 2^{96} = 2^{167} + 2^{160} \approx 2^{167} \quad . \tag{18}$$

Therefore the presented attack on Salsa20/5 has data complexity 2^7 chosen plaintexts and time complexity 2^{167} encryptions. As shown in Table 5, it is comparable to the attack proposed by Crowley [5].

Table 4. Estimating the probabilities of differentials for three rounds of Salsa20 using UNAF differences.

i	Δ_i^3	$\{\Delta^U\}_i^3$	$\hat{p}_{\text{add}}$	$\hat{p}_{\text{unaf}}/\#\Delta^U$	p_{exper}
9	`80000000`	`80000000`	$2^{-10.00}$	$2^{-4.00}$	$2^{-3.38}$
13	`ffe00100`	`00200100`	$2^{-15.75}$	$2^{-7.75}$	$2^{-4.93}$
14	`ff00001c`	`01000024`	$2^{-16.29}$	$2^{-8.31}$	$2^{-6.35}$
1	`00e00fe4`	`01201024`	$2^{-23.01}$	$2^{-13.04}$	$2^{-10.18}$
2	`00000800`	`00000800`	$2^{-35.59}$	$2^{-16.62}$	$2^{-11.08}$
3	`fff000a0`	`001000a0`	$2^{-41.48}$	$2^{-20.04}$	$2^{-14.68}$
6	`01038020`	`01048020`	$2^{-41.76}$	$2^{-21.91}$	$2^{-15.68}$
7	`ffefc000`	`00104000`	$2^{-44.65}$	$2^{-22.15}$	$2^{-17.42}$

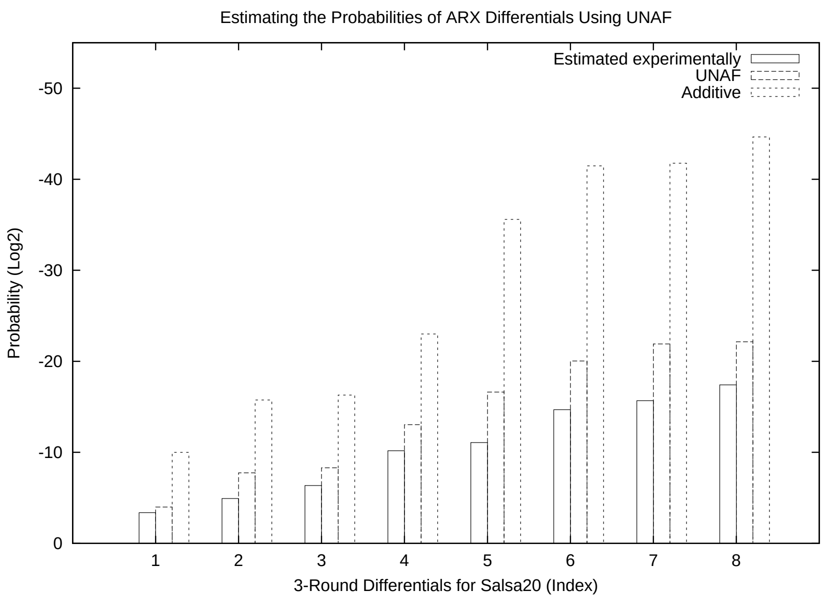


Fig. 4. Three estimates of the probabilities of eight differentials for three rounds of Salsa20, based on the data from Table 4: (1) estimation obtained experimentally, (2) based on UNAF differences and (3) based on single additive differences.

4 Conclusion

In this paper, we introduced UNAF differences. These are sets of specially chosen additive differences used to estimate the probabilities of differentials through sequences of ARX operations more accurately.

We presented the main UNAF theorem, which shows how a UNAF difference groups several possible additive differences together. Further, we investigated the propagation of UNAF differences through the ARX operation. We defined the UNAF differential probability of ARX and noted that it can be computed efficiently using the S-functions framework proposed by Mouha et al.

UNAF differences were applied to the cryptanalysis of the stream cipher Salsa20. We found that for three rounds of Salsa20, the probability of the best differential based on additive differences is estimated as 2^{-10}. Evaluating the same probability using UNAF differences leads to the value 2^{-4}. The latter is closer to the the probability of the differential $2^{-3.39}$ that was determined experimentally.

A general algorithm for the automatic search for differentials was briefly discussed. It was used to find high-probability UNAF differentials for three rounds of Salsa20. One of them was used to mount a key-recovery attack on Salsa20 reduced to five rounds. The attack has a time

Table 5. Overview of key-recovery attacks on Salsa20.

Rounds	Reference	Time	Data	Type of Differences
Salsa20/5	**Our result**	$\mathbf{2^{167}}$	$\mathbf{2^{7}}$	**Additive**
Salsa20/5	Crowley [5]	2^{165}	2^{6}	`XOR`
Salsa20/6	Fischer et al. [10]	2^{177}	2^{16}	`XOR`
Salsa20/7	Aumasson et al. [1]	2^{151}	2^{26}	`XOR`
Salsa20/8	Aumasson et al. [1]	2^{251}	2^{31}	`XOR`

complexity of 2^7 and a data complexity of 2^{167}. It therefore does not improve the best-known attack on the cipher. Nevertheless, to the best of our knowledge, this is the first cryptanalysis result on Salsa20 that is based on additive differences. Furthermore, we expect that the attack can be extended to more rounds. One possibility in this direction is to group two or more `ARX` operations and consider them as a single operation. Another is to improve the method for finding differential characteristics for multiple rounds.

The results in this paper were obtained for the Salsa20 stream cipher. We see the application of UNAF differences to other `ARX`-based ciphers as another interesting topic for future research.

Acknowledgments. The authors would like to thank Florian Mendel for his detailed comments and for his suggestions on the overall structure of the paper. His feedback was critical for improving the quality of the final text. We thank prof. Vincent Rijmen for reviewing the draft version of the paper. We also thank our colleagues at COSIC for the useful discussions, as well as the anonymous reviewers for their detailed comments.

References

1. J.-P. Aumasson, S. Fischer, S. Khazaei, W. Meier, and C. Rechberger. New Features of Latin Dances: Analysis of Salsa, ChaCha, and Rumba. In K. Nyberg, editor, *FSE*, volume 5086 of *Lecture Notes in Computer Science*, pages 470–488. Springer, 2008.
2. J.-P. Aumasson, L. Henzen, W. Meier, and R. C.-W. Phan. SHA-3 proposal BLAKE. Submission to the NIST SHA-3 Competition (Round 2), 2008.
3. D. J. Bernstein. The Salsa20 Family of Stream Ciphers. In M. J. B. Robshaw and O. Billet, editors, *The eSTREAM Finalists*, volume 4986 of *LNCS*, pages 84–97. Springer, 2008.
4. E. Biham and A. Shamir. Differential Cryptanalysis of DES-like Cryptosystems. *J. Cryptology*, 4(1):3–72, 1991.
5. P. Crowley. Truncated differential cryptanalysis of five rounds of Salsa20. SASC 2006 Workshop: Stream Ciphers Revisted. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/073, 2005. `http://www.ecrypt.eu.org/stream`.
6. J. Daemen and V. Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer, 2002.
7. N. M. Ebeid and M. A. Hasan. On binary signed digit representations of integers. *Des. Codes Cryptography*, 42(1):43–65, 2007.
8. eSTREAM. ECRYPT stream cipher project. `http://www.ecrypt.eu.org/stream`.
9. N. Ferguson, S. Lucks, B. Schneier, D. Whiting, M. Bellare, T. Kohno, J. Callas, and J. Walker. The Skein Hash Function Family. Submission to the NIST SHA-3 Competition (Round 2), 2009.
10. S. Fischer, W. Meier, C. Berbain, J.-F. Biasse, and M. J. B. Robshaw. Non-randomness in eSTREAM Candidates Salsa20 and TSC-4. In R. Barua and T. Lange, editors, *INDOCRYPT*, volume 4329 of *Lecture Notes in Computer Science*, pages 2–16. Springer, 2006.
11. P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions On Systems Science And Cybernetics*, 4(2):100–107, July 1968.
12. H. Lipmaa and S. Moriai. Efficient Algorithms for Computing Differential Properties of Addition. In M. Matsui, editor, *FSE*, volume 2355 of *LNCS*, pages 336–350. Springer, 2001.
13. H. Lipmaa, J. Wallén, and P. Dumas. On the Additive Differential Probability of Exclusive-Or. In B. K. Roy and W. Meier, editors, *FSE*, volume 3017 of *LNCS*, pages 317–331. Springer, 2004.
14. M. Matsui and A. Yamagishi. A New Method for Known Plaintext Attack of FEAL Cipher. In *EUROCRYPT*, pages 81–91, 1992.

15. N. Mouha, V. Velichkov, C. De Cannière, and B. Preneel. The Differential Analysis of S-Functions. In A. Biryukov, G. Gong, and D. R. Stinson, editors, *Selected Areas in Cryptography*, volume 6544 of *Lecture Notes in Computer Science*, pages 36–56. Springer, 2010.
16. G. W. Reitwiesner. Binary arithmetic. *Advances in Computers*, 1:231–308, 1960.
17. A. Shimizu and S. Miyaguchi. Fast Data Encipherment Algorithm FEAL. In *EUROCRYPT*, pages 267–278, 1987.
18. V. Velichkov, N. Mouha, C. De Cannière, and B. Preneel. The Additive Differential Probability of ARX. In A. Joux, editor, *FSE*, volume 6733 of *LNCS*, pages 342–358. Springer, 2011.

A Proof of Theorem 1

The following Lemma provides the condition under which the probability $\mathrm{adp}^{\oplus}$ is non-zero.

Lemma 1 (Theorem 2 of [13]). *All differences $\Delta^+ a$, $\Delta^+ b$ and $\Delta^+ c$ for which* $\mathrm{adp}^{\oplus}(\Delta^+ a, \Delta^+ b \rightarrow \Delta^+ c) > 0$*, are $\Delta^+ a = \Delta^+ b = \Delta^+ c = 0$, and*

$$\Delta^+ a = \Delta^+ a[n-1 \dots q+1] \parallel \Delta^+ a[q] \parallel 0^* \ , \tag{19}$$
$$\Delta^+ b = \Delta^+ b[n-1 \dots q+1] \parallel \Delta^+ b[q] \parallel 0^* \ , \tag{20}$$
$$\Delta^+ c = \Delta^+ c[n-1 \dots q+1] \parallel \Delta^+ c[q] \parallel 0^* \ , \tag{21}$$

where $\neg(\Delta^+ a[q] = \Delta^+ b[q] = \Delta^+ c[q] = 0)$ and $\Delta^+ a[q] \oplus \Delta^+ b[q] = \Delta^+ c[q]$. Each of the sub-word differences $\Delta^+ a[n-1 \dots q+1]$, $\Delta^+ b[n-1 \dots q+1]$ and $\Delta^+ c[n-1 \dots q+1]$ can take any arbitrary value. The symbol $*$ represents the Kleene star.

We proceed next with the proof of Theorem 1.

Proof. From Reitwiesner's algorithm for the construction of the NAF [16], it follows that if the first non-zero bit (starting from the LSB) of $\Delta^+ a_i$ is at position q, then the first non-zero bit of its NAF representation $\Delta^N a_i$ is also at position q. Since all $\Delta^+ a_i$ in (5) belong to the same UNAF set $\Delta^U a$, the first non-zero bit for all of them is in the same position q. The same observation holds for $\Delta^+ b_j$ and $\Delta^+ c_k$. From $\mathrm{adp}^{\oplus}(\Delta^+ a, \Delta^+ b \rightarrow \Delta^+ c) > 0$ and Lemma 1, it follows that $\Delta^+ a[q] \oplus \Delta^+ b[q] = \Delta^+ c[q]$. Therefore $\Delta^+ a_i[q] \oplus \Delta^+ b_j[q] = \Delta^+ c_k[q], \forall i, j, k$. Again by Lemma 1, it follows that if $\Delta^+ a$ is replaced by any $\Delta^+ a_i$ belonging to the same UNAF set $\Delta^U a$, the resulting probability $\mathrm{adp}^{\oplus}$ is still non-zero. The same observation can be made for $\Delta^+ b$ and $\Delta^+ c$, which completes the proof. □

B Computation of udp$^{\mathtt{ARX}}$

The probability $\mathrm{udp}^{\mathtt{ARX}}$ can be efficiently computed using the S-function framework [15, 18]. We briefly describe this computation below. It is also a part of a toolkit that will be made publicly available.

The propagation of input UNAF differences $\Delta^U a$, $\Delta^U b$ and $\Delta^U d$ to output UNAF difference $\Delta^U e$ is represented as an S-function. The latter is used to compute 16 adjacency matrices. Each of them corresponds to a given value of the i-th bit of each of the four UNAF differences and connects a set of possible input states to a set of possible output states.

The differential $(\Delta^U a[i], \Delta^U b[i], \Delta^U d[i+t] \xrightarrow{t} \Delta^U e[i+t])$ at bit position i is written as the bit string $w[i] \leftarrow (\Delta^U a[i] \parallel \Delta^U b[i] \parallel \Delta^U d[i+t] \parallel \Delta^U e[i+t])$. At each bit position $0 \leq i < n$, the index $w[i] \in \{0, \dots, 15\}$ selects one of the 16 adjacency matrices $A_{w[i]}$. The probability $\mathrm{udp}^{\mathtt{ARX}}$ is computed as follows:

$$\mathrm{udp}^{\mathtt{ARX}}(\Delta^U a, \Delta^U b, \Delta^U d \xrightarrow{t} \Delta^U e) = \sum_{j=0}^{14} L_j \left(\prod_{i=n-t}^{n-1} A_{w[i]} \right) R \left(\prod_{i=0}^{n-t-1} A_{w[i]} \right) C_j \ . \tag{22}$$

In (22), the summation is performed over each of the 14 possible initial states. The reason for having multiple initial states is the bit rotation by t positions, as explained in [18]. The multiplication by the projection matrix R at bit position t is necessary because of the rotation operation. The column vectors $C_j,\ 0 \leq j < 15$ represent the 15 possible initial states. The row vectors $L_j,\ 0 < j < 15$ represent their corresponding final states. For further details, we refer to [18].

Note that the matrices $A_{w[i]}$ are of dimension 540×540, but these can be minimized to 60×60 by combining equivalent states using the algorithm of [15, §3.5] .

C An Algorithm for Finding the Best Output Difference

Let $\Box$ be an operation that takes a finite number of n-bit input words $a_1, b_1, d_1, \ldots$ and computes an n-bit output word $c_1 = \Box(a_1, b_1, d_1, \ldots)$. Let $\bullet$ be a type of difference. Let α,β,ζ,... and γ be differences of type $\bullet$ such that $a_1 \bullet a_2 = \alpha$, $b_1 \bullet b_2 = \beta$, $d_1 \bullet d_2 = \zeta$, ... and $c_1 \bullet c_2 = \gamma$ for some a_2,b_2,d_2,... and some c_2. The differential probability with which input differences α, β, ζ, ... propagate to output difference γ with respect to the operation $\Box$ is denoted as $\bullet\mathrm{dp}^{\Box}(\alpha, \beta, \zeta, \ldots \to \gamma)$. Finally, let the difference $\bullet$ be such that it is possible to express its propagation through the operation $\Box$ as an S-function consisting of N states. Therefore, there exist adjacency matrices $A_{w[i]}$ such that the probability $\bullet\mathrm{dp}^{\Box}$ can be efficiently computed as $LA_{w[n-1]} \ldots A_{w[1]}A_{w[0]}C$, where $L = [1\,1 \cdots 1]$ is a $1 \times N$ matrix and $C = [1\,0 \cdots 0]^T$ is an $N \times 1$ matrix (as in [15]). The problem is to find an output difference γ such that its probability p_γ over all possible output differences is maximal:

$$p_\gamma = \bullet\mathrm{dp}^{\Box}(\alpha, \beta, \zeta, \ldots \to \gamma) = \max_j \ \bullet\,\mathrm{dp}^{\Box}(\alpha, \beta, \zeta, \ldots \to \gamma_j) \ . \tag{23}$$

We represent (23) as a problem of finding the shortest path in an *node-weighted binary tree.* We define the binary tree $T = (N, E)$, where N is the set of nodes and E is the set of edges. The height of T is $n + 1$ with a dummy start node positioned at level -1 and the leaves positioned at level $n - 1$. Each node at level $i : 0 \leq i < n$ contains a value of $\gamma[i]$, where $i = 0$ is the LSB and $i = n - 1$ is the MSB. Every node on level i has two children at level $i + 1$. Since the input differences $\alpha, \beta, \zeta, \ldots$ are fixed, at every bit position i we can choose between two matrices $A_{w[i]}$, corresponding to the two possibilities for the output difference $\gamma[i]$.

To find the output difference with the highest probability, we use the A* search algorithm [11]. In this algorithm, an evaluation function f can be computed for every node in the search tree. The f-function represents the weight of a node, and is based on the cost of the path from the start node, and a heuristic that estimates the distance to the goal node. The algorithm always expands the node with the highest f-value (corresponding to the highest probability). The A* search algorithm guarantees that the optimal solution will be found, provided that the evaluation function f never underestimates the probability of the best output difference. After introducing some definitions, we will define an evaluation function f and prove in Theorem 2 that this f satisfies the required condition.

Let vector $X_i = [x_{i,0}\, x_{i,1} \cdots x_{i,N-1}]$ be a transition probability vector, i.e. $x_{i,r} \geq 0$ for $0 \leq r < N$ and $\sum_{r=0}^{N-1} x_{i,r} \leq 1$. We define H_r as a column vector of length N, of which the r-th element (counting from 0) is 1 and all other elements are 0. The cost of a node at level i is then denoted by $\|X_i\|$ (the 1-norm of X_i) and is calculated as $\|A_{w[i]}A_{w[i-1]} \cdots A_{w[0]}C\|$. Let us define a sequence of row vectors $\hat{G}_{i,r},\ 0 \leq r < N$ and $0 \leq i < n$. Each $\hat{G}_{i,r}$ is a product of matrices $LA_{w[n-1]}A_{w[n-2]} \ldots A_{w[i+1]}$, where each of the A-matrices are chosen such that $\hat{G}_{i,r}H_r$ is maximized. The choice of the A-matrices may differ for different values of r. We define row vector G_i as the product of matrices $LA_{w[n-1]}A_{w[n-2]} \ldots A_{w[i+1]}$, where the A-matrices are chosen such that $G_i X_i$ is maximized. For a node at level i with cost $\|X_i\|$, the evaluation function f is defined as $\sum_{r=0}^{N-1} \hat{G}_{i,r} H_r x_{i,r}$.

Theorem 2. *The evaluation function $f = \sum_{r=0}^{N-1} \hat{G}_{i,r} H_r x_{i,r}$ never underestimates the probability of the best output difference.*

Proof. The following inequality holds: $\hat{G}_{i,r} H_r \geq G_i H_r$ for $0 \leq r < N$. The latter can be proven by contradiction: if $\hat{G}_{i,r} H_r < G_i H_r$ for some r, then $\hat{G}_{i,r}$ is not the product of A-matrices that maximizes $\hat{G}_{i,r} H_r$, which contradicts its definition. Because probabilities are non-negative, we can multiply both sides of the inequality by the state probability $x_{i,r}$, to obtain $\hat{G}_{i,r} H_r x_{i,r} \geq G_i H_r x_{i,r}$, $0 \leq r < N$. By summing the left and the right sides of the N inequalities, we obtain $\sum_{r=0}^{N-1} \hat{G}_{i,r} H_r x_{i,r} \geq \sum_{r=0}^{N-1} G_i H_r x_{i,r} = G_i X_i$. By definition, $G_i X_i$ is the best choice of A-matrices, starting from transition probability X_i. This proves that the left-hand side of the inequality never underestimates the probability, which proves the theorem. $\square$

Before we can apply the A* algorithm to compute the best output difference, we must determine the values of $\hat{G}_{i,r} H_r$ for $0 \leq i < n$ and $0 \leq r < N$. This is done by again running the A* algorithm for the most significant bit, then for the two most significant bits, and so on until we process the entire word. For the MSB, we define $\hat{G}_{n-1,r} = L$ for $0 \leq r < N$. For the two MSBs, we run the A* algorithm for every $0 \leq r < N$, setting the transition probability vector X_{n-2} to H_r. This allows us to compute $\hat{G}_{n-2,r} H_r$. This process is continued until $\hat{G}_{0,r} H_r$ for $0 \leq r < N$ is calculated. Having calculated all values of $\hat{G}_{i,r} H_r$, we then use the A* algorithm to search for the best output difference by setting the state transition probability vector $X_{-1} = C$. Pseudo-code of the entire A* search algorithm is provided in Algorithm 1.

D Attack on Salsa20/5 using UNAF Differences

Fig. 5 illustrates the attack presented in Sect. 3.3. Gray boxes denote guessed words and white boxes denote words that are either known or can be computed.

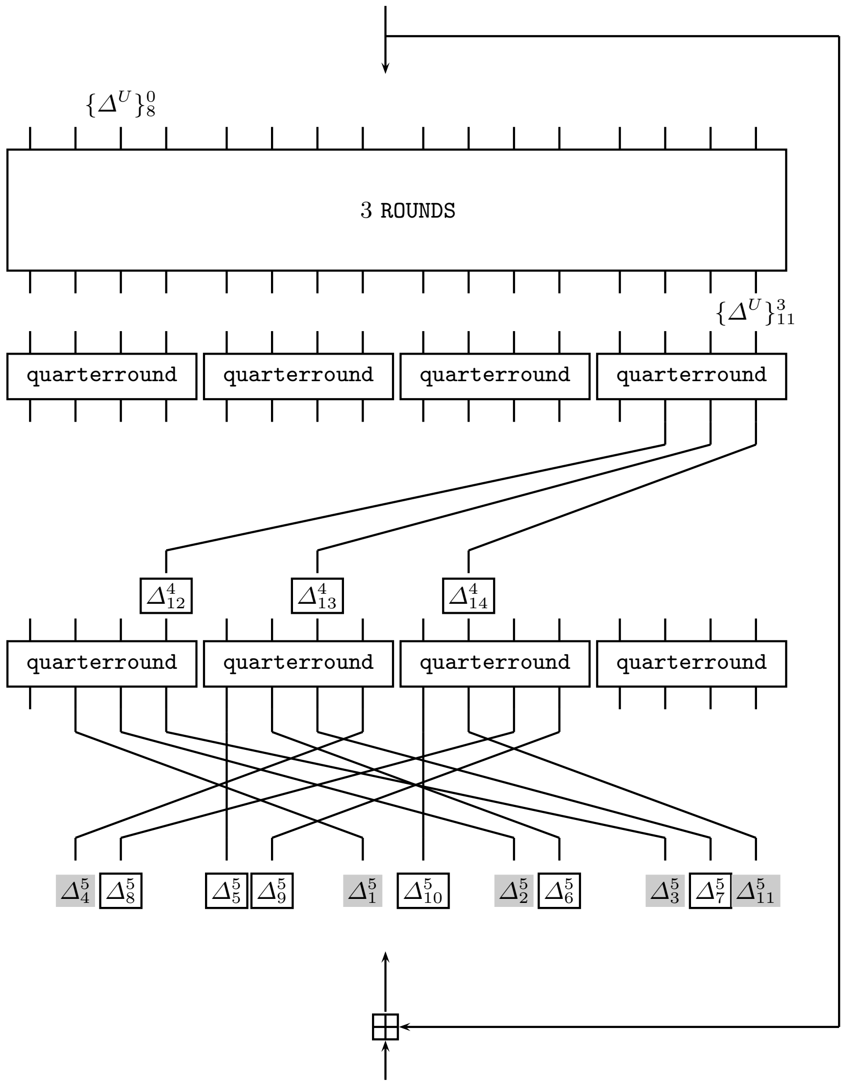


Fig. 5. Key-recovery attack on Salsa20/5 using the 3-round UNAF differential $\{\Delta^U\}^0_8 \rightarrow \{\Delta^U\}^3_{11}$. Gray boxes denote guessed words; white boxes denote words that are either known or can be computed.

Algorithm 1 Find the Best Output Diff. of Type $\bullet$ w.r.t. Operation $\Box$.

Input: Matrices $A_{w[i]}$ for $\bullet\mathrm{dp}^{\Box}$; input diffs. α, β, ζ, $\ldots$,; num. states N.
Output: Output difference γ and probability p_γ such that

$$p_\gamma = \bullet\mathrm{dp}^{\Box}(\alpha, \beta, \zeta, \ldots \to \gamma) = \max_j \ \bullet\mathrm{dp}^{\Box}(\alpha, \beta, \zeta, \ldots \to \gamma_j) \ .$$

```
1: Define struct node = $\{\mathrm{index},\ \gamma,\ f_{\mathrm{index}-1},\ \hat{H}_{\mathrm{index}-1}\}$
2: Init priority queue of nodes ordered by $f$: $Q = \varnothing$
3: Init output difference: $\gamma \leftarrow \varnothing$
4: for $i = n-1$ downto $0$ do
5:     if $i = n-1$ then
6:         $\hat{G}_i \leftarrow L = [1\ 1 \cdots 1]$
7:     else
8:         $\hat{G}_i \leftarrow [\hat{G}_{i,0}\ \hat{G}_{i,1} \ldots \hat{G}_{i,N-1}]$
9:     end if
10:    if $i = 0$ then
11:        $N = 1$
12:    end if
13:    for $r = 0$ to $N-1$ do
14:        Reset priority queue: $Q = \varnothing$
15:        Init the total probability of node $v_{i-1}$: $f_{i-1} \leftarrow 1$
16:        Init the transition probability vector $v_i$: $\hat{H}_{i-1} \leftarrow \hat{H}_{i-1,r}$
17:        Init node $v_i \leftarrow \{i,\ \gamma,\ f_{i-1},\ \hat{H}_{i-1}\}$
18:        Add new node to the queue: $Q$.push($v_i$)
19:        $v_{\mathrm{best}} \leftarrow Q$.top();  $\{j,\ \gamma,\ f_{j-1},\ \hat{H}_{j-1}\} \leftarrow v_{\mathrm{best}}$
20:        while $j \neq n$ do
21:            Remove $v_{\mathrm{best}}$ from the queue: $Q$.pop()
22:            for $q = 0$ to $1$ do
23:                Set the $j$-th bit of $\gamma$: $\gamma[j] \leftarrow q$
24:                Estimate the total probability: $f_j \leftarrow \hat{G}_j A^q_{w[j]} \hat{H}_{j-1}$
25:                Compute the transition probability vector: $\hat{H}_j \leftarrow A^q_{w[j]}\ \hat{H}_{j-1}$
26:                Init child of $v_{\mathrm{best}}$: node $v^q_{j+1} \leftarrow \{j+1,\ \gamma,\ f_j,\ \hat{H}_j\}$
27:                Add the child to the queue: $Q$.push($v^q_{j+1}$)
28:            end for
29:            Extract the node with the lowest total cost: $v_{\mathrm{best}} \leftarrow Q$.top()
30:            $\{j,\ \gamma,\ f_{j-1},\ \hat{H}_{j-1}\} \leftarrow v_{\mathrm{best}}$
31:        end while
32:        $v_{\mathrm{best}} \leftarrow Q$.top();  $f_{\mathrm{best}} \leftarrow$ get_cost($v_{\mathrm{best}}$)
33:        Set the $r$-th element of $\hat{G}_i$: $\hat{G}_{i,r} \leftarrow f_{\mathrm{best}}$
34:    end for
35: end for
36: Extract the node with highest total probability: $v_{\mathrm{best}} \leftarrow Q$.top()
37: Get the output difference associated to $v_{\mathrm{best}}$: $\gamma, p_\gamma \leftarrow$ get_gamma($v_{\mathrm{best}}$)
38: return $\gamma, p_\gamma$
```