Conditional Differential Cryptanalysis of the Post-Quantum ARX Symmetric Primitive Salsa20

Anaïs Querol Cruz

université PARIS DIDEROT PARIS 7

MPRI M2 Internship Report

Inria inventeurs du monde numérique

Conditional Differential Cryptanalysis of the Post-Quantum ARX Symmetric Primitive Salsa20

Anaïs Querol Cruz, supervised by María Naya Plasencia
SECRET team, Inria Paris

20 August 2018

General context

Symmetric key cryptography is an essential part of communication systems, where a secret key is used to protect data confidentiality. Surprisingly, the only way of trusting these ciphers is to perform continuous analysis that update the security margin. With the advent of quantum computers in an arguably near future, the security of nowadays ciphers has been put into question. While most currently used asymmetric primitives would be completely broken, doubling the key size of symmetric constructions provides the same level of security with respect to exhaustive key search. However, we still have a long way to go in the field of quantum cryptography and further cryptanalysis must be carried out to reassure the validity of these emerging ciphers.

Research problem

This internship has taken place in the context of the ERC project QUASYModo, which aims to provide quantum-secure symmetric primitives. We have studied the Salsa20 family of ciphers [Ber08b], which has received very little cryptanalysis ever since the most relevant result one decade ago [AFK+08]. Recent research has shown some evidence that 12-rounds Salsa suffice to provide security against the current best known differential attacks [CM16]. Despite their believed resistance against quantum computers, no one has ever performed cryptanalysis on ARX primitives with a quantum adversary in mind, due to the lack of theory surrounding them. Such scrutiny is utterly important now that the inclusion of this cipher suit in TLS 1.3 is almost complete.

Contribution

Before providing the first quantum attack we tried to improve the best classical ones. My contribution to the stated problem can be summed up in the following points. First, an introductory cryptanalysis of Salsa20/8 (section 3). This comprises a study of the diffusion of the cipher, a construction-based formula to find good differentials, an analysis of linearized versions to find neutral bits and a probabilistic heuristic to estimate differences of ARX rounds. Second, we propose a faster attack on 8 rounds of 256 bit key Salsa combining conditional cryptanalysis and the novel idea of forward PNBs (4.2). Third, we revisit some state-of-the-art attacks to give more correct time complexities (4.3).

Arguments supporting its validity

During this internship, we used the toolkit Merengue, a set of functions written in Python and C for the cryptanalysis of Salsa and other ARX ciphers. We programmed this software to design the proposed attack, thanks to the multiple functionalities provided. In order to guarantee its validity, the toolkit makes use of the official implementation of Salsa20 submitted to the eSTREAM portfolio[1]. Together with a test suite, this program is publicly accessible on the author's GitHub account[2].

[1] `www.ecrypt-eu.org/stream/e2-salsa20.html`
[2] `https://github.com/queroliita/merengue`

Summary and future work

This document gathers a brief view of the current panorama in the attacks on the symmetric cipher Salsa20, supplementary differential and linear cryptanalysis performed on this ARX primitive, and the introduction of a new cryptanalysis approach that lays the foundations of our attack on 256 bit key Salsa20/8 with time complexity $2^{241.8}$ and data $2^{31.7}$, which is the fastest known result of this cipher. We intend to submit these results to some important cryptologic conferences in early 2019.

The next question to be addressed is the feasibility to perform an attack on over 9 rounds of Salsa. We give some ideas, that we would like to pursue, that could help meet this target in the near future through imposing conditions on the attacker controlled initial bits. Another important step to be taken by this team is to tailor a quantum-based attack on ARX ciphers by adapting existing classical attacks in the quantum setting.

Notes and acknowledgements

The present document is written in English because of my insufficient proficiency in the French language. I must clarify in advance that the work presented in this report is not a quantum study. Owing to the lack of sufficient time, we focused on classical attacks and left quantum strategies for further partnership during my PhD studies.

I would like to acknowledge the *Fondation Sciences Mathématiques de Paris* for awarding me with the PGSM fellowship to study the MPRI. Special thanks to Anne Canteaut, who has performed an excellent mentorship during this year introducing me into the world of secret keys. My gratitude to María Naya Plasencia, for giving me the opportunity to work at this top research institute while making me feel at home.

Contents

1 Background

The focus of our research, the 256 bit key Salsa20 pseudorandom function, is explained in this section. This ARX primitive, namely add-rotate-xor, is the basis of the stream cipher of the same name. We also explain its expected security and a brief introduction to differential and linear attacks. When we cannot attack the whole cipher, cryptanalysis of modern primitives considers reduced rounds versions of the ciphers to determine the security margin, since they present more easily exploitable properties.

1.1 Salsa20 Cipher

The Salsa20 cipher was created by Daniel J. Bernstein as a candidate in the eSTREAM project [Ber05]. It is a 512-bit state stream cipher divided into 32-bit words represented in little-endian. This bitstring depends on a 256-bit key, a 128-bit constant, a 64-bit nonce and a 64-bit counter (note these last 128 bits of initial value are attacker-controlled in our quite realistic model). Because 4 words are fixed for all instances of the cipher, this pseudorandom function maps $\mathbb{F}_2^{384} \to \mathbb{F}_2^{384}$, with 32 bytes of unknown input. Its initial state can be seen as a square matrix of 16 words, following the structure below:

c_0	k_0	k_1	k_2
k_3	c_1	n_0	n_1
p_0	p_1	c_2	k_4
k_5	k_6	k_7	c_3

(a)

$c_0 = \texttt{61 70 78 65}$
$c_1 = \texttt{33 20 64 6E}$
$c_2 = \texttt{79 62 2D 32}$
$c_3 = \texttt{6B 20 65 74}$

(b)

$X =$

x_0	x_1	x_2	x_3
x_4	x_5	x_6	x_7
x_8	x_9	x_{10}	x_{11}
x_{12}	x_{13}	x_{14}	x_{15}

(c)

$X^T =$

x_0	x_4	x_8	x_{12}
x_1	x_5	x_9	x_{13}
x_2	x_6	x_{10}	x_{14}
x_3	x_7	x_{11}	x_{15}

(d)

FIGURE 1: Structure of a Salsa20 state: (a) Location of keywords, constants, nonces and counters in the initial state; (b) Constants in hexadecimal; (c) Odd rounds state; (d) Even rounds state.

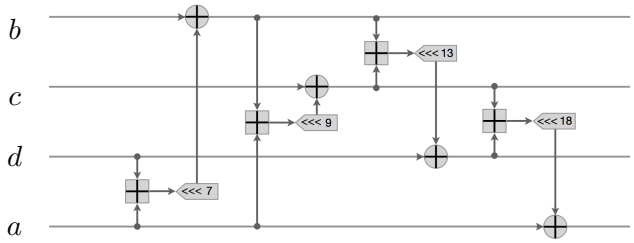
The underlying machinery of the cipher Salsa is an ARX function that modifies the initial state X to obtain the final state, referred to as X^R. This function called QuarterRound is applied R consecutive times on X, depending on the number of rounds specified in the mode of the cipher Salsa20/R: Salsa20/8 (already broken), Salsa20/12 (recommended) and Salsa20/20 (for added security).

a	d	c	b
b	a	d	c
c	b	a	d
d	c	b	a

(a)

$b \oplus= (a+d) \lll 7$
$c \oplus= (a+b) \lll 9$
$d \oplus= (c+b) \lll 13$
$a \oplus= (c+d) \lll 18$

(b)



(c)

FIGURE 2: Definition of a QuarterRound: (a) Location of word types; (b) ARX operations of a QuarterRound; (c) Circuit representation of a QuarterRound.

A Salsa20/R keystream $Z = (X + X^R)$ is computed after applying the QuarterRound function R times (the official versions being either 8, 12 or 20) to the initial state X, followed by a modular addition feedforward to prevent backward substitution. This pseudorandom function is applied to the columns of the matrix independently in odd rounds, whereas even rounds use its rows instead. Another way of putting it, with shorter software implementation, is to perform a matrix transposition after every QuarterRound, except for the last one. This function updates each word by XORing its previous value with the addition of the two words above, rotated to the left a predefined number of bits. The reverse Salsa function is usually referred to with negative exponents. That is, the initial state would be recovered if the operation $X = (Z - X)^{-R}$ were performed.

1.2 Expected Security

The ciphertext generated by an additive stream cipher (the most common type) results from XORing the keystream produced by the pseudorandom generator with the original message, recalling the one-time-pad. The main difference lies in the pseudorandom function used to generate the secret keystream in the former, from a small secret key of n bits (the seed). The goal of the cryptographer is to build a primitive that generates sequences undistinguishable from random, so that the best method to guess the n bits of the used seed is brute force.

This generic attack by brute force has a cost of 2^n computations, as recovering the good seed is always possible by trying them all, checking if the generated keystream is the same as a given one. This key-recovery attack should be the best one against the cipher. Otherwise, another key-recovery attack in symmetric cryptography is considered a break as long as it is faster than exhaustive search over the keyspace (*i.e.* $< 2^{256}$ for the studied version of Salsa).

The cryptanalysis results that we will explain throughout this document assume the model of known-plaintext attack (KPA). In this widespread model, the attacker is given both the plaintext and its corresponding ciphertext. In the present case, the attacker will have access to Z and Z'. Plus, we reasonably assume the attacker can control the nonce and the block counter.

1.3 Differential and Linear Cryptanalyisis

Differential cryptanalysis studies how an input difference between states can generate a certain difference in the output. We define now the particular case of single bit differentials, whose notation is easier to understand. We refer to input differential Δ_b^0 as the XOR between two states X, X' that differ in the bit b. After applying the primitive on both initial states, the output differential Δ_β^r is computed as the XOR of both final states Y, Y' in the bit β after r rounds. In the case of a perfectly random function 50% of the times $y_\beta = y'_\beta$. When the stream cipher uses a weak pseudorandom function, there may be some correlation between these two bits, meaning that the single bit input difference has some meaningful effect on the β values of Y, Y'. More formally, $\Pr\{y_\beta \neq y'_\beta | x_b \neq x'_b \wedge x_i = x'_i \ \forall i \neq b\} = \frac{1}{2}(1+\varepsilon)$ for a non-negligible bias ε. Usually, a bias is considered significant if the number of samples used to compute this probability is $N > \varepsilon^{-2}$.

Multibit differentials can be classified in three groups: first order multibit differentials, where the XOR difference of multiple bit positions in the output is measured after fixing a single bit difference in the input; multiple order single bit differentials, where a number of differences in the initial states may produce a bias in a single output bit; and multiple order multibit differentials, which considers multiple bits both in the input and the output. Generally speaking, they comprise the cases where the initial states differ in *in* bits and the output difference is measured in some bits *out*. Similarly, $Pr\{\bigoplus_{\beta \in out} y_\beta \neq y'_\beta | \bigwedge_{b \in in} x_b \neq x'_b,\ x_i = x'_i \ \forall i \neq b\} = \frac{1}{2}(1+\varepsilon)$.

Given a number of independent binary random variables $X_1, \ldots, X_n$ with $\Pr\{X_i = 0\} = \frac{1}{2}(1+\varepsilon_i)$ each, the piling-up lemma [Mat94] states that the bias of the probability that all of them hold can be computed as the product of each individual bias: $\Pr\{\sum_1^n X_i = 0\} = \frac{1}{2}(1 + \prod_1^n \varepsilon_i)$. This lemma is used extensively to estimate linear approximations of stream ciphers.

2 State of the Art

The strategies presented in this section aim at recovering the secret key. Two years after Bernstein presented Salsa20 in 2005, Aumasson, Fischer, Khazaei, Meier and Rechberger published the first attack over 8 rounds of this cipher in 2^{251} time with 2^{31} data. Eversince then, very little progress has been made to decrease the attack complexity of Salsa20/8, which remains the furthest broken cipher in the family (see Table 1 for details). This section presents such attack, which has become a widespread base technique to perform the cryptanalysis of Salsa.

Rounds	Year	Reference	Time	Data	Rounds	Year	Reference	Time	Data	Memory
5	2005	[Cro06]	2^{165}	2^{6}	7	2012	[SZFW13]	2^{148}	2^{24}	
5	2006	[FMB+06]	2^{81}	2^{24}	7	2017	[CM17]	2^{137}	2^{61}	
5	2006	[SZFW13]	2^{55}	2^{10}						
5	2006	[CM17]	2^{8}	2^{8}	8	2007	[AFK+08]	2^{251}	2^{31}	
					8	2012	[SZFW13]	2^{250}	2^{27}	
6	2006	[FMB+06]	2^{177}	2^{15}	8	2015	[MGM15]	$2^{247.2}$	$2^{27.2}$	
6	2006	[SZFW13]	2^{73}	2^{16}	8	2016	[Mai16]	$2^{245.5}$	$2^{22.5}$	
6	2006	[CM17]	2^{32}	2^{32}	8	2017	[CM17]	$2^{244.9}$	$2^{30.8}$	
7	2007	[TSK+07]	2^{184}	2^{12}	8	2008	[PSB08]	2^{192}	2^{191}	2^{192}
7	2007	[AFK+08]	2^{153}	2^{26}	8	2008	[Ber08a]	2^{96}	2^{64}	2^{96}

TABLE 1: Chronology of previous 256-bit Salsa20 cryptanalysis

2.1 Aumasson's Attack

In [AFK+08], Aumasson et al. presented the best known attacks so far to this cipher family, *i.e.* Salsa, ChaCha and Rumba. Taking advantage of the similar construction of the first two ciphers, they designed an attack working for both of them which allows to break 256-bit version of Salsa20/8 using truncated forward differentials and probabilistic backward computation. In short, they find an initial bit that is correlated with another bit after 4 rounds and they measure this correlation performing 4 reverse rounds from the output state of Salsa20/8 guessing fewer keybits than with exhaustive search.

Forward

Because Salsa20 is a pseudorandom function, one may expect it will not present a perfectly random behaviour for the first rounds. The forward step of this attack consists on finding a good single bit input differential $\mathcal{ID}$ that produces some high bias a few rounds later in the output differential $\mathcal{OD}$, generating an interesting pair $(\mathcal{OD}|\mathcal{ID})$. If the only difference between the initial states X and X' is the j^{th} bit of word i, we define the $\mathcal{ID}$ as $\Delta^0_{i,j} = x_{i,j} \oplus x'_{i,j} = 1$ and the function $f = \Delta^r_{p,q} = x^r_{p,q} \oplus x'^r_{p,q}$ outputs the XOR difference of the bit (p, q) between the two intermediate states X^r and X'^r, then the bias ε_f of the $\mathcal{OD}$ after $r < R$ rounds is defined by the probability over all nonces, n, and counters, p, $\Pr_{n,p}\{f = 1|\Delta^0_{i,j}\} = \frac{1}{2}(1 + \varepsilon_f)$, for the same fixed key, which is measured in the bit (p, q).

Unsurprisingly, the higher the bias the faster to detect and thus, the easier the attack. Their experiments found no significant bias for single bit differences for over 4 rounds. Aumasson et al. use the pair $(\Delta^4_{1,14}|\Delta^0_{7,31})$ with forward bias $\varepsilon_f = 0.131$, though many other better single differentials were found afterwards [MGM15] by exhaustive search. Other approaches suggest multibit higher order differentials [Ish12, SZFW13, CM17], sometimes obtaining higher biases after 5 rounds. However, none suffices to perform an attack over 8 rounds by mere forward computation.

Backward

Aumasson's R-round attack combines an r-round forward differential with $R - r$ inverse Salsa rounds and by guessing keybits. Given that an output state is defined as $Z = (X + X^R)$ where the state X was created using the seed k, an intermediate state can be computed as $X^r = (Z - X)^{r-R}$.

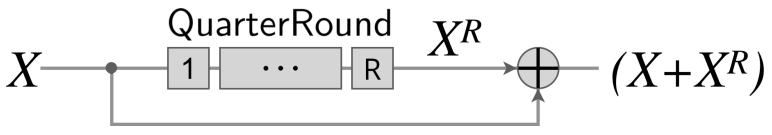


FIGURE 3: Scheme of the construction of a Salsa20/R keystream

The idea is to perform an exhaustive search over the m most influential keybits, reverse 4 rounds and expect a random outcome when the guess of keybits is incorrect. Put it another way, let $\widehat{X}$ be an initial state with the same diagonal constants and IV as X and key $\widehat{k}$, there will be no bias if there were no relation between Z and $\widehat{X}$. That is, if $\widehat{X}$ was built with a different choice of the key $\widehat{k} \neq k$, the outcome of $(Z - \widehat{X})$ will be a random matrix. Afterwards, an independent second search is used to obtain the remaining $256 - m$ keybits.

In order for this attack to work, the authors evaluate a function g over only m keybits and bias ε_g, which is expected to behave similarly to f such that $\Pr_{n,p}\{f = g\} = \frac{1}{2}(1 + \varepsilon_g)$. The backward function is defined as $g = y_{p,q} \oplus y'_{p,q}$ where $Y = (Z - \widehat{X})^{R-r}$ and $Y' = (Z' - \widehat{X'})^{R-r}$. It can be obtained by finding the most significant keybits and setting the n remaining ones to zero, the so called probabilistic neutral bits (PNBs). They use a neutrality measure such that $\frac{1}{2}(1+\gamma_i)$ is the probability that complementing the keybit k_i will not change the output of f when reversing by $R - r$ rounds from the output keystream. Fixing a threshold determines a compromise between the set of significant keybits ($\leq \gamma$) and the overall complexity of the attack: the higher the threshold, the slower the search but higher the bias as well.

$$\gamma_i \begin{cases} 1, & f \text{ never depends on } k_i \\ 0, & f \text{ half times depends on } k_i \\ -1, & f \text{ always depends on } k_i \end{cases}$$

FIGURE 4: Neutrality measure

Complexity

The actual attack would measure a total bias ε which can be estimated as $\varepsilon_f \cdot \varepsilon_g$. The authors use Neyman-Pearson's decision theory [Sie85] to estimate the number of input pairs satisfying the $\mathcal{ID}$ needed to detect such bias as $N \approx (\frac{\sqrt{\alpha \log 4}+3\sqrt{1-\varepsilon^2}}{\varepsilon})^2$, where $\Pr\{\text{false alarm}\} = 2^{-\alpha}$, for each choice of the subkey. When a non-negligible bias is detected, an additional exhaustive search is performed to obtain the remaining non-significant keybits. This means the attack time complexity is $2^m(N + 2^n 2^{-\alpha}) = 2^m N + 2^{256-\alpha}$ and data complexity N. Their experiments estimate an attack complexity of 2^{251} with 2^{31} pairs and $\alpha = 8$. They detect a bias $|\varepsilon_f^\star| = 0.131$, and using the threshold $\gamma = 0.12$, they obtain $|\varepsilon_g^\star| = 0.0011$, $|\varepsilon^\star| = 0.00015$ and $n = 36$ PNBs at $x^4_{1,14}$.

Seven years later, Maitra et al. revisited this attack [MGM15] using slightly modified parameters to obtain two better attacks with $2^{247.2}$ time complexity. They tried both increasing the median bias $|\varepsilon^\star|$ and the number of PNBs. The former strategy consists on taking the median of a few average measurements of ε to obtain a bias of 0.00060 instead and setting $\alpha = 12.82$ as well. Regarding the latter, they decrease the threshold γ down to 0.0488 and choose 41 PNBs with $|\varepsilon^\star| = 0.000106$.

2.2 Maitra's Approaches

After revising Aumasson's proposal, Maitra presented an attack on Salsa20/8 which outperformed the then fastest strategies. One year later, and together with Choudhuri, he published a new differential-linear cryptanalysis on Salsa, the most recent work on this cipher. We will describe now these two attacks, as well as the theoretical bases of the latter.

Chosen IV

The attacker in [Mai16] takes advantage of its control over the initial values to slow down the propagation of differences along the initial rounds of Salsa. In particular, the single $\mathcal{ID}$ in $\Delta^0_{7,31}$ produces between 4 and 22 differences after the first QuarterRound. Maitra's idea consists of choosing pairs whose x_7 will only generate 4 differences at this stage in order to measure a higher forward bias.

Column operations are performed independently, so one can just focus on the fourth column to understand the source of disparity between the number of differences. The first step does not depend on x_7, so there are no differences between x_3^1 and $x_3'^1$. The XOR in the second step keeps one single difference in the MSB of x_7^1. The addition in the third step will not produce a carry, so the only difference will occur at $\Delta_{11,12}^1$. Last, the number of differences at x_{15}^1 depends on the carry in the 12^{th} bit of $(x_7^1 + x_{11}^1)$. If the bits 11...0 do not produce a carry and $x_{7,12}^1 = 0$, then no differences will be propagated further. The same is true when there is carry and $x_{7,12}^1 = 1$. This means, in half of the cases only two more differences will occur at $\Delta_{15,17}^1$ and $\Delta_{15,30}^1$.

The attack itself follows the same steps than Aumasson's one, except for the proper choice of IVs. His experiment comprises 256 random keys, and for each fixed key he computes the median biases using the 2^{31} values for x_7^0 that produce only 4 differences after the first QuarterRound (being found by exhaustive search). Using 33 PNBs, he obtained $\varepsilon_f^\star = 0.228538$, $\varepsilon_g^\star = 0.013778$ and $\varepsilon^\star = 0.003154$. If $\alpha = 15$ then $N \approx 2^{22.5}$ so the overall attack complexity is $2^{245.5}$.

Hybrid Model

The analysis in [CM16] lays the foundations of the attack in the following section and gives an upper bound on the number of Salsa rounds needed to achieve the desired level of security. The proof is based on a linear version of the cipher which substitutes modular additions to exclusive OR operations ($+ \mapsto \oplus$). By the piling-up lemma, they state that any actual forward bias is upperbounded by the bias in the linear counterpart. This proof of security uses two well known results in cryptanalysis. First, a distinguisher requires the forward bias be more than $2^{-\frac{k}{2}}$ [MS02]. Otherwise, attacks cannot be faster than exhaustive search. Last, a feasible attack must satisfy $\varepsilon_f \varepsilon_g < 2^{-\frac{n}{2}}$.

The starting point of this proof is the number of bit dependencies between words in consecutive rounds. Because of the cipher structure, the first types of words will be the ones with the least number of dependencies with the previous round. This means b words have 3 dependencies, c words have 5, d words have 9 and a words have 15. Note the transpose between rounds swaps the roles of b and d words. Denoting $|\varepsilon_{\mathcal{OD}}^r| < 1 - \delta$ the forward bias in the output differential after r Salsa rounds, then we can compute an upper bound of this bias for each type of word using the dependencies above. This means, the biases after one more round can be estimated with the following expressions.

Bit	b^{-1}	c^{-1}	d^{-1}	a^{-1}	Bias upper bound
b_i	1	0	1	1	$\lvert\varepsilon_b^{r+1}\rvert < (1-\delta)^3$
c_i	1	1	1	2	$\lvert\varepsilon_c^{r+1}\rvert < (1-\delta)^5$
d_i	2	1	3	3	$\lvert\varepsilon_d^{r+1}\rvert < (1-\delta)^9$
a_i	3	2	4	6	$\lvert\varepsilon_a^{r+1}\rvert < (1-\delta)^{15}$

TABLE 2: Inter round word dependencies

With that in mind, one can check that the highest bias after 2 more rounds will be $|\varepsilon_b^{r+2}|$, upperbounded by $(1-\delta)^{27=1\cdot 9+0\cdot 5+1\cdot 3+1\cdot 15}$. Setting this quantity to 2^{-128} implies non-indistinguishability of the bias. One can stop after $r+2$ rounds if the forward bias after r rounds is less than 0.037402, higher than the best known $|\varepsilon_f^\star|$ for 5 Salsa rounds. Considering now the backward direction, it is known that $|\varepsilon_g| < 1$ for -4 rounds and single bit $\mathcal{OD}$ and obviously $n < k$, so the condition above will hold because $\varepsilon_f < 2^{-\frac{k}{2}}$ is already verified. Plus, their experiments show that no PNBs can be found for more than 4 rounds backwards. Put all together, this study gives a proof of security under differential attacks on a linearized version of the cipher after 12 rounds (*i.e.* 5+2+5). Given that the actual biases will be smaller than their linear version, it gives evidence that 256-bit key Salsa20/12, the accepted eSTREAM candidate, suffices to obtain the desired level of security with respect to this type of attacks. Conversely, it leaves an open door to the cryptanalysis of 9, 10 and 11 rounds of Salsa, which have anyhow no known improvement.

Multibit Differentials

The full version [CM17] of the paper above includes a differential-linear attack on Salsa20/8 that takes advantage of the structure of the cipher to find multibit differentials. Unlike single bit differentials, this task is unfeasible by brute force search (namely, in the order of $\binom{512}{x}$). Their approach gives a theoretical reason of many multibit differentials found in the literature.

Differential-linear analysis [BDK02] gives the means to study a cipher from a differential that creates a linear approximation. This version will output the same result as the original function with some biased probability $\Pr\{f = l\} = \frac{1}{2}(1 + \varepsilon_l)$, where the differential-linear bias is $\varepsilon_f \varepsilon_l^2$.

The Salsa QuarterRound is linear ($\varepsilon_l = 1$) when the addends of the ARX operation are the LSB, so there is no input carry and additions behave like XOR. Here, the bias of some active bits at round r can be expressed as a linear combination of active bits from the next round.

$$\varepsilon\{c_9^r\} \cdot \varepsilon\{a_0^r\} \approx \varepsilon\{c_9^{r+1} \oplus b_0^{r+1}\} \qquad \varepsilon\{d_{13}{}^r\} = \varepsilon\{d_{13}^{r+1} \oplus c_0^{r+1} \oplus b_0^{r+1}\} \qquad \varepsilon\{a_{18}^r\} = \varepsilon\{a_{18}^{r+1} \oplus d_0^{r+1} \oplus c_0^{r+1}\}$$

For all other cases ($\varepsilon_l < 1$), the linear approximation leads to some loss of information. They substitute modular additions $s_i = \alpha_i + \beta_i$ for $s_i = \alpha_i \oplus \beta_i \oplus \alpha_{i-1}$, which holds true 75% of the times. They extend this for $r + 2$ rounds to obtain multibit forward biases for up to 6 rounds. Experiments show that the highest biases can be observed when the $\mathcal{OD}$ is a d word. In particular, they find a single bit $\mathcal{ID}$ multibit differential of 19 $\mathcal{OD}$ after 6 rounds, whose equations follow this pattern:

$$\begin{aligned}
\varepsilon_l = \tfrac{1}{2^3} :\ b_{i+7}{}^r &= b_{i+7}^{r+1} \oplus a_i^{r+1} \oplus a_{i-1}^{r+1} \oplus d_{i-18}{}^{r+1} \oplus c_{i-18}^{r+1} \oplus d_i{}^{r+1} \oplus c_{i-13}^{r+1} \oplus b_{i-13}^{r+1} \oplus b_{i-14}^{r+1} \\
\varepsilon_l = \tfrac{1}{2^2} :\ c_{i+9}{}^r &= c_{i+9}^{r+1} \oplus b_i^{r+1} \oplus a_i^{r+1} \oplus a_{i-1}^{r+1} \oplus d_{i-18}{}^{r+1} \oplus c_{i-18}^{r+1} \\
\varepsilon_l = \tfrac{1}{2} :\ d_{i+13}{}^r &= d_{i+13}{}^{r+1} \oplus c_i^{r+1} \oplus b_i^{r+1} \oplus b_{i-1}^{r+1} \\
\varepsilon_l = \tfrac{1}{2} :\ a_{i+18}^r &= a_{i+18}^{r+1} \oplus d_i^{r+1} \oplus c_i^{r+1} \oplus c_{i-1}^{r+1} \\
\varepsilon_l = \tfrac{1}{2^6} :\ d_{13}^r &= b_0^{r+2} \oplus \overleftarrow{c}_{19}{}^{r+2} \oplus \overline{\overline{d}}_{19}{}^{r+2} \oplus \overline{\overline{d}}_{18}{}^{r+2} \oplus c_0^{r+2} \oplus \overleftarrow{d}_{23}{}^{r+2} \oplus \overline{\overline{a}}_{23}{}^{r+2} \oplus \overline{\overline{a}}_{22}{}^{r+2} \oplus \vec{b}_5{}^{r+2} \oplus c_5^{r+2} \\
&\quad \oplus d_{13}{}^{r+2} \oplus \overleftarrow{a}_6{}^{r+2} \oplus \overleftarrow{a}_5{}^{r+2} \oplus \overline{\overline{b}}_{20}{}^{r+2} \oplus \vec{c}_{20}{}^{r+2} \oplus \overline{\overline{b}}_6{}^{r+2} \oplus \vec{c}_{25}{}^{r+2} \oplus d_{25}^{r+2} \oplus d_{24}^{r+2}
\end{aligned}$$

Unfortunately, a large number of $\mathcal{OD}$ decreases enormously the number of PNBs. Then, the actual attack uses the linear approximation of d to go 5 rounds forwards and 3 rounds backwards with 42 PNBs. They use the multibit differential $(\Delta_{9,0}^5 \oplus \Delta_{13,0}^5 \oplus \Delta_{1,13}^5 | \Delta_{7,0}^0)$ with $\varepsilon_f = -0.114$. In order to get higher biases (double it, in fact), they also apply the Chosen IV strategy to ensure the minimal number of differences after the first round.

They perform an exhaustive computation on the last column to identify the suitable combinations of x_3, x_7, x_{11}, so the attack has a data complexity overhead of 2^{96}. With $\varepsilon_f = -0.233918$, $\varepsilon_g = 0.000752$, $\varepsilon = -0.000178$ and $\alpha = 15.5$, they get time complexity of $2^{244.9}$ and $N \approx 2^{30.8}$ (2^{96} in the worst case). Despite the smaller time complexity, its augmented data requirement makes this approach a trade-off, in comparison with the previous best known attack.

3 Tentative Cryptanalysis

This section shows the steps I made throughout this internship, as well as a basic iterative technique to estimate the bias introduced by an $\mathcal{ID}$ in linear time in the word length named PDARX.

3.1 Preliminary View

During the first part of my internship, I analysed simplified versions of Salsa for a better understanding of its behaviour. The most relevant remark is that the cipher structure itself promotes the presence of good differentials that include, what we called, "slow-update" and "slow-propagate" words.

Diffusion

Due to this cipher's construction, we observe words with different properties. First, words in the main diagonal have the largest number of dependencies on the words from the last round. Second, words of type b have fewer dependencies on words from last round. Third, words of type c are the slowest to be propagated (*i.e.* affect other words) Fourth, words of type d are the slowest to be updated (*i.e.* be affected by other words). One must notice that a and c words maintain the same role from odd to even rounds, whereas b and d words swap [CM16]. A priori, this means that a good differential may place its $\mathcal{ID}$ bit in a c word whereas one will observe the $\mathcal{OD}$ in a slow-update word b or d (after odd or even rounds, respectively).

Salsa20 takes 3 rounds until every word affects each word in the block, and 4 rounds until each bit affects every single bit. We checked the later statement considering linear dependencies with the propagation module of Merengue, that we will explain in section 4.1. The former claim can be easily proven by a careful study of the QuarterRound, counting the number of word dependencies. Thanks to the cipher structure, the behaviour of any word can be expressed by shifting the corresponding terms of any other column.

$\overleftarrow{a}$	d	$\vec{c}$	$\overline{\overline{b}}$
$\overleftarrow{b}$	a	$\vec{d}$	$\overline{\overline{c}}$
$\overleftarrow{c}$	b	$\vec{a}$	$\overline{\overline{d}}$
$\overleftarrow{d}$	c	$\vec{b}$	$\overline{\overline{a}}$

(a)

$\overline{\overline{c}}$	$\overline{\overline{c}}\leftarrow$	$\overline{\overline{c}}$	$\overline{\overline{c}}$
$\overleftarrow{c}$	$\overleftarrow{c}$	$\overleftarrow{c}\circ$	$\overleftarrow{c}$
c	c	c	$c\rightarrow$
$\vec{c}=$	$\vec{c}$	$\vec{c}$	$\vec{c}$

(b)

FIGURE 5: Our state notation: (a) Word types in odd rounds (b) Free words after two rounds

Trivially, the words in the initial state only depend on themselves. After the first round, b^1 word positions depend on 3 initial words a^0, b^0, d^0, whereas the remaining words in the column depend on the 4 initial words from the same column. Even from this early step, we can notice the uneven pattern between words. On the one hand, b words take longer to be updated with the same number of words. On the other hand, c words take longer to affect all words. After the transpose and second round, d^2 words have 11 word dependencies whereas all other words have 15. After the third round, each word depends on all initial words. These equations show the exact dependencies at $r = 2$:

$$
\begin{aligned}
d^2 &\Rightarrow d^1 + \overleftarrow{a}^1 + \overline{\overline{b}}^1 \Rightarrow \{ \quad a + b + c + d + \vec{a} + \vec{b} + \vec{c} + \vec{d} + \overline{\overline{a}} + \overline{\overline{b}} + \overline{\overline{d}} \}^0 \\
c^2 &\Rightarrow c^1 + \overleftarrow{d}^2 + \overline{\overline{a}}^1 \Rightarrow \{ \overleftarrow{a} + \overleftarrow{b} + \overleftarrow{c} + \overleftarrow{d} + a + b + c + d + \vec{a} + \vec{b} + \vec{d} + \overline{\overline{a}} + \overline{\overline{b}} + \overline{\overline{c}} + \overline{\overline{d}} \}^0 \\
b^2 &\Rightarrow b^1 + \overleftarrow{c}^2 + \overline{\overline{d}}^2 \Rightarrow \{ \overleftarrow{a} + \overleftarrow{b} + \overleftarrow{c} + \overleftarrow{d} + a + b + d + \vec{a} + \vec{b} + \vec{c} + \vec{d} + \overline{\overline{a}} + \overline{\overline{b}} + \overline{\overline{c}} + \overline{\overline{d}} \}^0 \\
a^2 &\Rightarrow a^1 + \overleftarrow{b}^2 + \overline{\overline{c}}^2 \Rightarrow \{ \overleftarrow{a} + \overleftarrow{b} + \overleftarrow{d} + a + b + c + d + \vec{a} + \vec{b} + \vec{c} + \vec{d} + \overline{\overline{a}} + \overline{\overline{b}} + \overline{\overline{c}} + \overline{\overline{d}} \}^0
\end{aligned}
$$

Matrix in Figure 5(b) shows graphically the location of the free initial words after two rounds for each word cell. This means the rows do not depend on $\overline{\overline{c}} = x_7^0$, $\overleftarrow{c} = x_8^0$, $c = x_{13}^0$ and $\vec{c} = x_2^0$, from the first to the fourth. That is, slow-update words {12,1,6,11} do not depend at all on c words {2,7,8,13}, respectively, so this can become a bias in higher rounds. Apart from that, d cells do not depend on the columns corresponding to the left position. This means, {12,1,6,11} words do not depend on {(3,7,11,15), (0,4,8,12), (1,5,9,13), (2,6,10,14)}, each.

Given that the attacker can only set the $\mathcal{ID}$ on the IV in our reasonable model, one can theoretically infer good differentials of the form $(\Delta^r_{1,q}|\Delta^0_{7,j}), (\Delta^r_{6,q}|\Delta^0_{8,j})$ in a first approach. The literature in differential cryptanalysis of Salsa20 shows a clear majority of differentials following the above structure (see Table 3 for details). The research held in [MGM15] validates the important role of slow-update words, verifying experimentally that they form an input/output differential cycle after 4 rounds.

Order	$\mathcal{OD}$	$\mathcal{ID}$	Bias	Reference
(1|1)	$(\Delta^4_{1,14}$	$\Delta^0_{7,31})$	0.131	[AFK+08]
(1|1)	$(\Delta^4_{6,26}$	$\Delta^0_{7,31})$	0.201	[SZFW13]
(1|1)	$(\Delta^4_{6,26}$	$\Delta^0_{7,31})$	0.195	[MGM15]
(1|1)	$(\Delta^4_{11,26}$	$\Delta^0_{8,31})$	0.191	[MGM15]
(1|1)	$(\Delta^4_{11,17}$	$\Delta^0_{8,31})$	0.164	[MGM15]
(2|1)	$(\Delta^4_{1,0} \oplus \Delta^4_{2,9}$	$\Delta^0_{7,26})$	-0.600	[AFK+08]
(2|1)	$(\Delta^4_{6,23} \oplus \Delta^4_{7,0}$	$\Delta^0_{8,17})$	0.178	[Ish12]
(3|1)	$(\Delta^5_{9,0} \oplus \Delta^5_{13,0} \oplus \Delta^5_{1,13}$	$\Delta^0_{7,0})$	-0.114	[CM17]
(1|2)	$(\Delta^4_{1,7}$	$\Delta^0_{7,24} \wedge \Delta^0_{8,17})$	0.670	[SZFW13]
(3|2)	$(\Delta^5_{9,0} \oplus \Delta^5_{13,0} \oplus \Delta^5_{1,13}$	$\Delta^0_{7,17} \wedge \Delta^0_{8,23...29})$	≈ 0.31	[CM17]
(1|4)	$(\Delta^5_{6,1}$	$\Delta^0_{2,8} \wedge \Delta^0_{6,12} \wedge \Delta^0_{14,19} \wedge \Delta^0_{14,31})$	7.7e-4	[FMB+06]

TABLE 3: Some high biased differentials of 256-bit Salsa20

Linearization

In order to understand Aumasson's differential $(\Delta^4_{1,14}|\Delta^0_{7,31})$, we created a naive Salsa20 linear version by simply substituting modular addition by XOR operation $(+ \mapsto \oplus)$, wich only holds when no input carries are produed. Certainly, diffusion became slower because of linearity, obtaining a large number of neutral bits. We realised that the word x_7 generated the least number of dependencies on the word x_1 after 4 rounds. Moreover, the bit $z_{1,j+15}$ presented the largest number of dependencies on the bit $x_{7,j}$, naively resulting in Aumasson's good differential for $j = 31$. Nonetheless, this differential only showed 54 non-affecting keybits, whereas the pair $(\Delta^4_{6,j+15}|\Delta^0_{8,j})$ presented 58 and thus it is a candidate for becoming a better differential. The reason for placing the $\mathcal{ID}$ on the MSB is to avoid propagation of differences to left bits after the initial round.

One can profit from the XOR non-idempotency to increase the number of actual neutral bits. The advantage of this kind of linear version relies on the ease to compute the final value of one bit by xoring operators appearing an odd number of times. Thus, input bits acting an even number of times (including no occurrences at all) will be neutral bits. After running this experiment for 4 rounds, we noticed x_1 would only have 95 forward neutral keybits whereas the word with the largest number of neutral keybits is x_3 with 132. The full state can be observed in Figure 6(a).

Our next linear approximation consisted on replacing operations $s_i = \alpha_i + \beta_i$ by $s_i = \alpha_i \oplus \beta_i \oplus \alpha_{i-1}$, $\forall i \neq 0$, which holds with probability 75%, as recently suggested in [CM17]. This approach treats LSB differently, so one cannot expect a constant number of intraword neutral bits as previously. Instead, we show the bits with the largest (Figure 6(b)) and smallest (Figure 6(c)) numbers of neutral keybits after considering odd occurrences only.

Despite computing forward neutral bits of a linear version and not PNBs, it helps understanding the diffusion in Salsa. When obtaining the number of non-occurring bits in this approximation, we noticed the neutral bits of the bits within a word follow the same structure, except for a few bits with extra neutral bits. Predictably, slow-update and slow-propagate words are the only ones with actual neutral bits after 4 forward rounds. The matrix in Figure 6(d) shows the maximum number of neutral keybits, being $x_{11,7}$ the highest.

In order to improve the attack, one needs both a high biased differential and a large number of neutral keybits. In spite of the hypothetical $(\mathcal{OD}|\mathcal{ID})$ pair $(\Delta^4_{6,j+15}|\Delta^0_{8,j})$, one can see that the word x^4_1 may indeed have a larger number of PNBs. Let t the type of the current word, $\vec{t}$ refers to the word of type t on the column on the right, $\overleftarrow{t}$ on the left and $\overline{\overline{t}}$ two columns beyond. Appendix B. Table 7 shows the location of these neutral bits in the linearized setting.

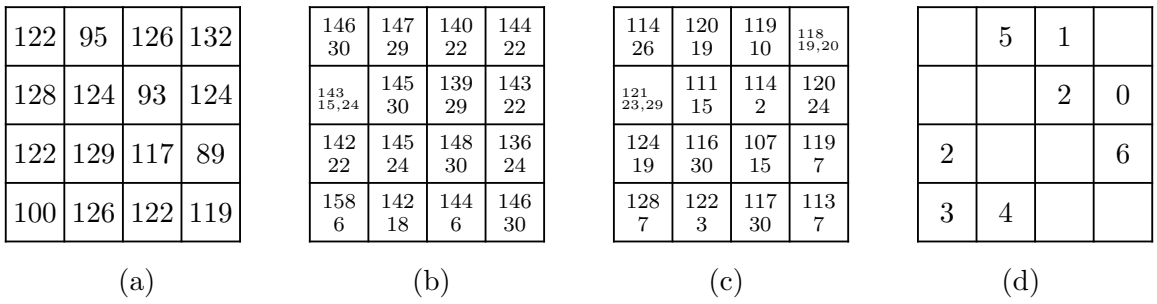


FIGURE 6: Neutral bits after linearization experiments

Remarks

The above Salsa linearized versions comprise a clear example of how easily breakable linear ciphers can be. Some known results show AR systems are theoretically equivalent to their ARX counterpart, whereas XR and AX systems can always be broken [KN10].

Modular addition is a non-linear operation because the carries from less significant bits will affect the output value. In particular, when adding two 32-bit words modulo 2^{32}, the output carry bit in the MSB is ignored, whereas the value of the LSB is always the XOR of the addends. Similarly, modular addition behaves linearly until the first appearance of the bit addends $1+1$ starting from the LSB.

Given a Salsa20 ciphertext $Z = (X + X^R)$, one can always compute the half of X corresponding to the words in the diagonal and the IV by simply subtracting its known values as $x_i^R = z_i - x_i$. Given an output state with even R, one can compute the whole 10^{th} word from the previous round as $x_{10}^{R-1} = x_{10}^R \oplus (x_8^R + x_9^R)_{\lll 18}$. Plus, one can control the exact value to be xored to x_{14}^0 in the first round because the attacker knows $(x_6^0 + x_{10}^0)_{\lll 7}$. Since the last bit of the constant term is 0, the value of $x_{14,8}^1$ will be computed using the linear operation $(x_{6,1}^0 \oplus 1 \oplus x_{14,8}^0)$.

3.2 Probabilistic Differential ARX

The role of the different operations of ARX ciphers is well known: the non-linearity of modular addition brings confusion, rotation offers intra-word diffusion and XOR gives inter-word diffusion and linearity. It is a general belief that the combination of these operations provides certain level of security after a sufficiently high number of rounds. However, their exact effect is not so well understood because of the lack of theory concerning ARX constructions. For this reason, we built our own basic heuristic that helps us understand the propagation of differences and approximate some bias after a number of Salsa rounds, which can be extended to any ARX cipher.

Prior work has been conducted in order to compute differential properties of addition. The differential probability of addition $\mathrm{DP}^+(\alpha, \beta \mapsto \gamma)$ is defined as the probability that $(x + y)$ differs from $((x \oplus \alpha) + (y \oplus \beta))$ at the positions set by γ. The fastest algorithm for the average case is logarithmic in the bitstring length [LM02]. Besides, the additive differential probability of ARX, namely $\mathrm{adp}^{\mathrm{ARX}}(\Delta\alpha + \Delta\beta, \Delta\lambda \xrightarrow{r} \Delta\gamma)$, is defined as the proportion of pairs $(a + b, d)$ such that the difference between the two outputs equals some fixed value after the ARX operation $((a + b) \lll r) \oplus d$. An algorithm to compute this probability was proposed in [VMDCP11], with linear time complexity.

We present a technique called probabilistic differential addition, which on input two vectors of differences, computes the probabilities of outputting a different bit value after performing a modular addition on the words satisfying such differentials. Together with differential XOR and rotation, it can be used to determine the probabilities of outputting a different bit value after performing one ARX operation. Besides the naive algorithm to compute these probabilities in exponential time, we propose a linear time approach to address this problem.

Notation

Let $\{0,1\}$ be the possible binary values of one bit, we can represent the uncertainty of the value of one bit as some real number between 0 and 1. For instance, if $b = 0.5$ then the actual value of b will be 0 or 1 with equal probability. Similarly, let $\{\mathbf{0},\mathbf{1}\}$ be the possible values of the difference between a pair of bits, we can represent the uncertainty of the value of the difference between the two bits as some real number between $\mathbf{0}$ and $\mathbf{1}$. This means $\mathbf{0}$ translates to two possible values, $b = 0 \wedge b' = 0$ or $b = 1 \wedge b' = 1$; and $\mathbf{1}$ implies $b = 0 \wedge b' = 1$ or $b = 1 \wedge b' = 0$. Thus, $\mathbf{0.5}$ means a pair of bits will differ from each other half of the time.

Applying a constant rotation on a word is as simple as rotating to the left the vector of values. Computing the XOR of two bitwords can be done using Venn diagrams as $\texttt{XOR}(A,B) = A+B-2\cdot\texttt{AND}(A,B)$ where $\texttt{AND}(A,B) = A \cdot B$. These two ARX operations are valid for both vectors of values and vectors of differences. However, addition must be treated differently.

In the case of modular addition with valued inputs we use a 32-bit full adder. This means the sum is computed iteratively from the LSB to the left and initial carry $c_0 = 0$ such that $\texttt{ADD}(a_i,b_i,c_i) = \texttt{XOR}(a_i,b_i,c_i)$ with output carry c_{i+1} where $\texttt{OR}(A,B) = A + B - \texttt{AND}(A,B)$. That is, there will be a carry when at least two inputs are valued 1.

In order to compute modular additions with differential inputs, it is possible to analyse each bit case separately and extend them to the general case (detail in Table 8). Let $\texttt{ADD}(\mathbf{a_i},\mathbf{b_i},\mathbf{c_i}) = (\mathbf{s_i},\mathbf{c_{i+1}})$, then $\Pr\{\neq \text{value}_i\} = \mathbf{s_i}$ and $\Pr\{\neq \text{carry}_{i+1}\} = \mathbf{c_{i+1}}$. If any input bit value is fixed and known beforehand, as in the case of diagonal constants, the propagated uncertainty in can be reduced when the differential addends are flipped and there is no difference in the input carry.

$$\texttt{ADD}\begin{cases} (\mathbf{0},\mathbf{0})^{\mathbf{c}} &= (\frac{\mathbf{1}}{\mathbf{2}})^{\frac{\mathbf{c}}{\mathbf{2}}} \\ (\mathbf{0},\mathbf{1})^{\mathbf{c}} &= (\mathbf{1}-\mathbf{c})^{\frac{\mathbf{1}}{\mathbf{2}}} \\ (\mathbf{1},\mathbf{0})^{\mathbf{c}} &= (\mathbf{1}-\mathbf{c})^{\frac{\mathbf{1}}{\mathbf{2}}} \\ (\mathbf{1},\mathbf{1})^{\mathbf{c}} &= \mathbf{c}^{\frac{\mathbf{1}-\mathbf{c}}{\mathbf{2}}} \end{cases} \qquad \texttt{ADD}\begin{cases} (\mathbf{0},\mathbf{1})^{\mathbf{0}} &\mapsto (0/0,\mathbf{1})^{0/0} = (\mathbf{1}-\mathbf{c})^{\mathbf{0}} \\ (\mathbf{0},\mathbf{1})^{\mathbf{0}} &\mapsto (1/1,\mathbf{1})^{1/1} = (\mathbf{1}-\mathbf{c})^{\mathbf{0}} \\ (\mathbf{1},\mathbf{0})^{\mathbf{0}} &\mapsto (\mathbf{1},0/0)^{0/0} = (\mathbf{1}-\mathbf{c})^{\mathbf{0}} \\ (\mathbf{1},\mathbf{0})^{\mathbf{0}} &\mapsto (\mathbf{1},1/1)^{1/1} = (\mathbf{1}-\mathbf{c})^{\mathbf{0}} \end{cases}$$

Heuristic

There exists a simple but exhaustive method to obtain the modular addition of differential inputs, *i.e.* computing the addition over all possible inputs satisfying such differences and counting the number of pairs with different output bits among all combinations.

By the law of total probability, this quantity can be calculated in linear time in the word size. This is, for each triplet of input bits $(\mathbf{a_i},\mathbf{b_i},\mathbf{c_i})$ we can compute the probabilistic output value and its probabilistic output carry as follows:

$$\texttt{ADD}\begin{cases} \mathbf{s_i} &= (1-\mathbf{a_i})(1-\mathbf{b_i})\frac{\mathbf{1}}{\mathbf{2}} + (1-\mathbf{a_i})\mathbf{b_i}(\mathbf{1}-\mathbf{c_i}) + \mathbf{a_i}(1-\mathbf{b_i})(\mathbf{1}-\mathbf{c_i}) + \mathbf{a_i}\mathbf{b_i}\mathbf{c_i} \\ \mathbf{c_{i+1}} &= (1-\mathbf{a_i})(1-\mathbf{b_i})\frac{\mathbf{c_i}}{\mathbf{2}} + (1-\mathbf{a_i})\mathbf{b_i}\frac{\mathbf{1}}{\mathbf{2}} + \mathbf{a_i}(1-\mathbf{b_i})\frac{\mathbf{1}}{\mathbf{2}} + \mathbf{a_i}\mathbf{b_i}\frac{\mathbf{1}-\mathbf{c_i}}{\mathbf{2}} \end{cases}$$

This approach can estimate the bias generated by an $\mathcal{ID}$, so it helps us finding good differentials. More interestingly, we use this technique to observe the propagation of differences along successive Salsa rounds. In particular, we evaluated this heursitic with $x^0_{7,31} = \mathbf{1}$ and all remaining input bits to $\mathbf{0}$, specifying the constants values. After running 4 rounds of differential Salsa, we observed a high biased differential output value at $x^4_{1,14} = \mathbf{0.48675}$, hence this intermediate value is more likely to be equal. Likewise, when imposing the single bit $\mathcal{ID}$ at $x^0_{8,31} = \mathbf{1}$, we observed the same biases provided $x_1 \mapsto x_6$. The reason why this procedure does not output the actual bias is the lack of linkage between bits. Nonetheless, it is still useful to understand the source of differences between states.

4 Contribution

This section includes our contribution to the cryptanalysis of Salsa20. We revisited existing papers to formulate a new attack over Salsa20/8 in time $2^{241.8}$ and data $2^{31.7}$ using conditional differential cryptanalysis 4 rounds forwards and 4 rounds backwards. This method constitutes a factor $2^{3.1}$ time improvement over the state-of-the-art fastest attack [CM17] (but $2^{0.9}$ data overhead). More importantly, we provide new directions to exploit in order to further improve the attacks. We also introduce Merengue, a toolkit we developed for the cryptanalysis of this cipher.

4.1 Merengue Toolkit

Over the course of this internship, we developed a toolkit for cryptanalysis of ARX ciphers, and more specifically, the Salsa20 primitive. It is divided in two parts, depending on the desired kind of analysis. Because of the different features of programming languages, the first module contains an intuitive Python set of functions for simplified versions of the cipher whereas the second module consists of a powerful kit programmed in C to compute the actual biases of Salsa for the offline attack.

Python Kit

This kit uses Python lists to represent the cipher state, that is, a list of length 16 where each element is a list of 32 bits, which will take between 1 and 4 bytes of memory (characters $\{'0','1'\}$, integers in the set $\{0,1\}$ or floating point numbers $\{0.0,1.0\}$). Despite its slow treatment of data, Python's high level programming style gives the user a comfortable syntax to visualize the matrix. Plus, it provides a number of functions to check some properties of linearized versions of Salsa. The following paragraphs present, in general lines, the different modes used in this software.

This kit counts with eight basic functions modelling Salsa: `QuarterRound()`, `transpose()`, `swap()`, `column()`, `ARX()`, `ADD()`, `ROT()`, `XOR()`. They all can work in different modes depending on the specific measurement. It includes some functions that initialize the state: `bitstring()`, `diagonal()`, `inivalue()`, `keywords()`.

This software provides two cipher modes. The encryption mode `"enc"` of the function `Salsa()` returns the keystream after a number of rounds (with or without feedforward) and the value of an output differential between two initial states. The `"inv"` mode of the function `Aslas()` returns an intermediate state X^r by applying reverse Salsa rounds over the state X^R or $Z = (X + X^R)$.

The kit can handle linearized versions of Salsa. Given an $\mathcal{ID}$, the `"pro"` mode of `propagate()` outputs the linear propagation of this $\mathcal{ID}$ assuming the substitution $(+ \mapsto \oplus)$. The $\mathcal{ID}$ is set to 1 and the output list will gather the number of times that this $\mathcal{ID}$ affects the resulting equation of each output bit. Making use of this function, `LNBs()` returns the coordinates of the linearly neutral bits after r Salsa rounds with respect to an $\mathcal{OD}$. The `"lin"` mode of `linearize()` computes the state after a number of rounds when modular additions are substituted by the linear expression $\alpha[i] \oplus \beta[i] \oplus \alpha[i-1]$.

The program includes the function `unroll()` that computes dependencies from backward direction. Its `"xor"` mode returns the coordinates of all initial bits that affect linearly an output bit (p, q) after a number of rounds. It keeps track of the whole equation of such bit recursively, simplifying its terms whenever possible. Conversely, `"lin"` performs this same operation when the linearization comes from the linearization with probability $\frac{3}{4}$.

On another note, the function `carrify()` with mode `"car"` computes probabilistic output values of the Salsa cipher. It initializes the state with floating point numbers following this convention: for all attacker known bit values, these positions are set to 0.0 or 1.0 accordingly; keybits are initialized to 0.5 to represent perfect randomness. After a few rounds, if the values of the output state do not tend to $\frac{1}{2}$, then the cipher beneath has very little confusion.

Similarly, the function `pdarx()` with mode `"pda"` applies the PDARX heuristic to estimate the expected difference between two initial states after a number of Salsa rounds. This means the matrix is initialized with **0.0** except for the positions of the (possibly non-single) $\mathcal{ID}$, which are set to **1.0** to represent the XOR difference of these bits. Unlike the above convention, a value equal to **0.5** implies two bits will be different or equal with same probability. Our experiments show a slower tendency to 0.5 in the PDARX function, resulting in a more helpful tool for cryptanalysis. In any case, the most realistic tool and thus the most convenient to attack Salsa is explained in the next section.

C Library

This compact library written in C leverages its low-level capabilities to achieve a much higher performance for faster cryptanalysis of Salsa20. This implementation performs in average 2^{24} QuarterRound functions per second, which is to say, applying 8 rounds of Salsa to 2^{21} initial states each second. We verified the correct functioning of this program reproducing existing results from the literature in attacks to Salsa. Besides, it uses the official code of this primitive underneath.

This program is equipped with the primitive `salsa()`, the inverse round function `alsas()` and a number of useful measurements. This toolkit contains the function `getbias()`, which can be run to compute the forward bias ε_f with parameter `"Ef"`, the backward bias ε_g with `"Eg"`, the total bias ε with `"E"`. Also, we can obtain probabilistic neutral bits with `getPNBs()`, both forwards `"for"` and backwards `"back"`. The test suite provided allows the user to invoke these functions with any configuration, by writing to `stdin` (the command line terminal). We can choose any of the above measurements, a valid single $\mathcal{ID}$ and (optional) multibit $\mathcal{OD}$, the number of samples (which includes the number of random initial values and different keywords, to compute the median values), the number of forward rounds r and total rounds R. More interestingly, the user can decide whether or not functions will constraint the initial values. They can receive up to two different kinds of conditions. First, maximizing determinism through the initial rounds of Salsa by controlling certain bits of the nonce or counter. Second, fixing forward probabilistic significant bits of the IV. Such strategies will be explained in depth in the following subsection.

Now, we give some tips for the desired functioning of this program. When using Aumasson's differential, the user can impose conditions on $x_{7,12}$, $x_{6,10}$ or none writing the configuration `CIV` as `7`, `6` or `n`, respectively. In the case of the dual differential, the user can condition on $x_{8,12}$ writing `8` (or none, as above). Regarding backward PNBs, it is possible to constraint the number of keybits by setting `BPNB` to `36`, `37` or `38` for Aumasson's differential with $\gamma = 0.12$ or `48` with $\gamma = 0.04$, and `30` or `31` for the dual. Forward PNBs can be controlled for Aumasson's differential with the parameter `FPNB` set to `47`, `52`, `54`. For more advanced settings, the file `testsuite.c` can be modified accordingly.

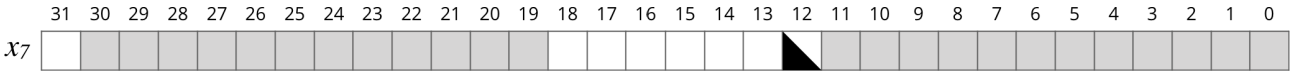
4.2 Conditional Differential Cryptanalysis of Salsa20

We applied conditional differential cryptanalysis, the technique presented in [KMNP10], to the standard attack by Aumasson to control some IV bits so as to delay the uncertainty imposed by the differential along the initial rounds of Salsa. We kindly suggest the reader running Merengue for a better understanding of the proposed attack. Its imported header gives precomputed structures to be used for a faster execution of the recommended settings.

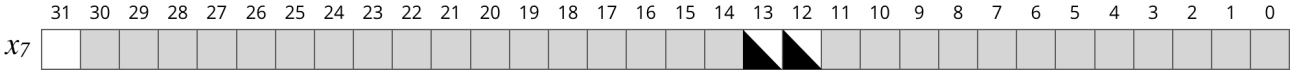
Conditioned IV

The ideas presented in this section are related to the work done in [Mai16]. Applying the ideas of conditional differential cryptanalysis to the choice of nonce and counter, we can find higher biases after 4 rounds. Picking the MSB for the $\mathcal{ID}$ is not arbitrary. Instead, it ensures slower propagation of differences, hence higher biases. Plus, one can fix some input bits to grant some properties will hold. The goal of this part is to explain an attacker strategy that exploits their control over the choice of concrete values of some bits of the IV.

The number of forward rounds for which it is possible to observe a good differential depends on the speed of the diffusion of the cipher. One can check that an Aumasson's $\mathcal{ID}$ generates up to 22 differences after the first QuarterRound, as already explained. Maitra's attack uses an exhaustive search over the word x_7^0 to choose the exact values that will only generate 4 differences, for a specific key. However, we noted such a large search space is not necessary. In fact, some bits of this nonce word do not influence the number of differences. The only IV bits determining the presence of carry at $(x_{11}^1 + x_7^1)_{12}$ are $x_{7,11\ldots0}^0$ due to x_7^1 and $x_{7,30\ldots19}^0$ because of x_{11}^1. Then, the search space could be reduced from 2^{32} to 2^{24}, and still be able to get the 2^{31} valid values for the IV, using any combination of bits for the remaining 7 neutral positions and 0 or 1 at $x_{7,12}^0$. Note the $\mathcal{ID}$ is a free bit.



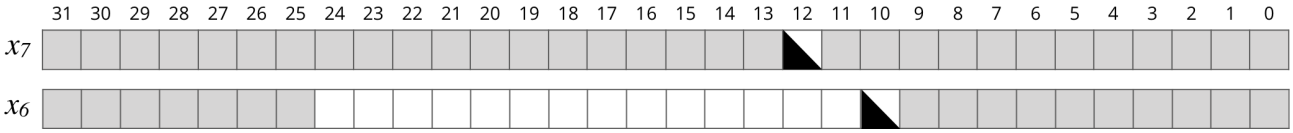
Besides, there is no need to keep track of all these IVs. For any combination of fixed key and non-free IV bits, either 0 or 1 at $x_{7,12}^0$ will produce only 2 differences at x_{15}^1. We cannot predict which bit value will be the correct one because the actual value of $x_{7,12}^1$ depends on the unknown key. What we can do is run the $\mathcal{ID}$ experiment twice and measure the forward bias after 4 rounds. The thread with the highest $|\varepsilon_f^\star|$ will have the correct bit guess, without exhaustive search at all, but using techniques from [KMNP10] instead. To prevent from storing the biases for each thread, another option consists on fixing $x_{7,18\ldots13}^0$ as well and find the correct bit value for $x_{7,12}^0$, which will be valid for all samples with the same key.



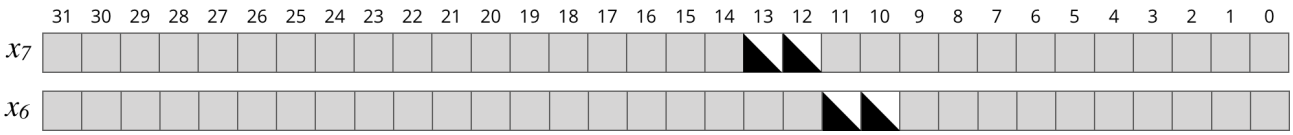
Nonetheless, the sharp reader will agree that sometimes the flipped bit $x_{7,12}^0$ can produce a different carry in each thread, leading to a different bit value in $(x_{3,19}^1 + x_{7,19}^1)$ and thus in $x_{11,0}^1$ between the two threads (but equal inside X and X'). Plus, this could result in a different carry value in $(x_{7,12}^1 + x_{11,12}^1)_{\lll 18}$ producing additional differences in x_{15}^1. This happens in the non-negligible case that the thread with $x_{11,0}^1 = 1$ sums to a large trail of zeroes in $(x_{7,11\ldots0}^1 + x_{11,11\ldots0}^1)$ and the thread with $x_{11,0}^1 = 0$ sums to a large trail of ones. The first case will produce a carry in the 12^{th} position whereas the second will not. In order to be sure that the measurements always correspond to a state producing 2 differences in x_{15}^1, we need to think of a way to control this undesired situation. Instead, we can also flip $x_{7,13}^0$ to stop this propagation of differences. In this case, there will be 4 threads, with only one of them leading to the highest bias.

Even if only 4 differences occur at round 1 and a higher bias can be detected, the 2 differences in the diagonal position will still propagate significant uncertainty shortly. We considered further control over the IV to reduce the number of differences after 2 rounds as well. The first ARX operation, where x_{15}^1 is added to x_{14}^1, is crucial to this aim.

In the spirit of the paragraph above, we want to control $x_{14,17}^1$ to grant at most 3 differences go to x_{12}^2 at $\Delta_{12,24}^2$, $\Delta_{12,5}^2$ and sometimes $\Delta_{12,6}^2$. In this sense, we do not care much about $\Delta_{15,30}^1$ because it only generates one more difference half of the times. Hence, we need to flip the bit $x_{6,10}^0$ and control the bits that could affect its input carry (*i.e.* the bits $16\ldots0$ of the addends x_{14}^1 and x_{15}^1). These are the bits $x_{6,31\ldots25}^0$, $x_{6,10\ldots0}^0$, and the whole x_7^0. In this case, for a fixed combination of keys and non-free bits at x_6^0 and x_7^0, we will have 4 independent runs of the experiment depending on the bit values of $x_{7,12}^0$ and $x_{6,10}^0$: (0,0), (0,1), (1,0), (1,1). The one with the best forward bias will satisfy both 4 differences after the first round and 2 or 3 at x_{12}^2.



As in the case of a single conditioned IV word, we can fix the whole x_7 and x_6 for the same key for a faster evaluation of biases. Then we flip $x^0_{7,13}$ and $x^0_{6,11}$ as well, so that the same concrete values of the conditioned bits hold for that key. During the first iteration of the loop, we will check the number of differences in x^1_{15} to get the correct values of $x^0_{7,13}$ and $x^0_{7,12}$. Afterwards, we do likewise in x^2_{12} to obtain the combination of $x^0_{6,11}$ and $x^0_{6,10}$. This way, there is no need to have 2 or 4 threads of execution to estimate these biases. Nonetheless, we must keep them during the actual attack, which will multiply the overall time complexity by the number of threads.



We measured the median forward bias with 2^{10} different keys and 2^{24} nonces and values for each key. If no conditions are imposed on the IV, we found Aumasson's expected bias of 0.1314. In the conditioned $x^0_{7,12}$ we found Maitra's expected bias of $|\varepsilon^\star_f| = 0.228$. Now, adding the condition at $x^0_{6,10}$ results in a forward bias of 0.252594. It is worth saying one can apply the first conditioned strategy to the differential $(\Delta^4_{6,14}|\Delta^0_{8,14})$ in a similar way (*i.e.* flip $x^0_{8,12}$ and fix $x^0_{8,30...19\_11...0}$) to obtain 2 differences at x^1_0. Here, we found a forward bias of 0.142761 for this differential and $|\varepsilon^\star_f| = 0.264526$ when fixing one condition. However, the second approach cannot be reproduced because we cannot control any bits of x^1_3. Plus, controlling more bits of the IV leaves very few free bits to perform our measurements.

On another note, if we perform PDARX on these conditioned scenarios, we will observe a higher number of differential outputs significantly different from **0.5**. Note the same biases observed when $\mathcal{ID} = \Delta^0_{7,31}$ can be measured if $\Delta^0_{8,31}$, provided $x_1 \mapsto x_6 \mapsto x_{11}$ (because of the structure of the cipher). Annex B. Table 9 shows such biases after 4 rounds of differential Salsa and $\mathcal{ID} = (\,\overline{\overline{c}}, 31)$.

Backward PNBs

Aumasson's attack leverages the existance of probabilistic neutral keybits to perform the backward stage of their attack. As a reminder, let X, X' be two initial states satisfying the $\mathcal{ID}$, Z, Z' the keystreams output by Salsa20/R and $\overline{X}, \overline{X'}$ the resulting input states after flipping the bit b_i, then this bit is a backward PNB if it does not affect the output of the function $f(Z, Z')$. That is, b_i is a non-significant keybit if the bias of $\Pr\{(X^4_{\mathcal{OD}} \oplus X'^4_{\mathcal{OD}} \oplus Y_{\mathcal{OD}} \oplus Y'_{\mathcal{OD}} = 0 | \Delta^0_{\mathcal{ID}} = 1)\}$ is higher than a threshold for the neutrality measure, where the states Y, Y' are computed by reversing $R - r$ Salsa rounds using the flipped states as $Y = (Z - \overline{X})^{R-r}$ and $Y' = (Z' - \overline{X'})^{R-r}$.

$(\Delta^4_{1,14}\|\Delta^0_{7,31})$	Keyword	Intraword bits	Keyword	$(\Delta^4_{6,14}\|\Delta^0_{8,31})$
	x_{11}	20	x_{12}	
	x_3	7, 8	x_4	
		13, 14, 31	x_{11}	
	x_{13}	18, 19, 20	x_2	
	x_4	24, 25, 26		
	x_1	26, 27, 28, 29, 30, 31		
	x_{14}	0, 1, 18, 19, 20, 21, 22, 23	x_3	
	x_{12}	5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17	x_1	

TABLE 4: BPNBs of dual differentials

We analysed this concept and detected a curious pattern on backward PNBs. When the differential is a slow-propagate/slow-update pair, we observed the positions of the BPNBs follow cycles. In particular, we found 63 of them when $\mathcal{OD} = (d, 14)$, as detailed in Table 10 (considering all kinds of bits, not only keybits). We checked Aumasson et al.'s 36 BPNKBs for $(\Delta^4_{1,14}|\Delta^0_{7,31})$ where $\overline{\overline{c}} = 7, d = 1$ and also obtained the BPNKBs for our hypothetically good differential $(\Delta^4_{6,14}|\Delta^0_{8,31})$. Unfortunately, because of the structure of the Salsa cipher, the latter only has 30 BPNKBs. Despite the slightly higher forward bias of this differential, this result suggests an overall slower attack.

The attack in [Mai16] pays very little attention to probabilistic neutral bits. In fact, they state "*we need to discard the PNBs corresponding to the keywords k_2, k_4. This reduces 3 PNBs (...) from the list of 36 PNBs*", but he never explains why. We ourselves have run the experiment a number of times using 2^{24} pairs for each keybit to understand this decision. Surprisingly, we checked that the number of BPNBs increases when we consider conditioned x_7. As a consequence of the overall rise of the biases in the conditioned scenario, neutrality measures are slightly higher and a larger number of bits exceed the threshold. Anyway, the observed backward bias decreases due to the existing tradeoff.

Experiments with Merengue show 64 BPNBs for this kind of differential pair with $\gamma = 0.12$, resulting in 37 BPNKBs for Aumasson's differential and 31 when $\overline{\overline{c}} = 8, d = 6$. The extra keybit corresponds to the bit $(\vec{b}, 2)$. It is worth noting the presence of actual neutral bits in the conditioned scenario, such as $(\vec{d}, 5)$ with bias $\gamma_i = 1.000$. Plus, when we consider conditions on both x_7 and x_6, we obtain another BPNKB at position $(\vec{b}, 12)$, resulting in a total number of 38. This is great news, because an increased number of BPNBs will partly compensate the overhead due to multiple threads with conditioned IV.

Once we get these BPNBs, we build function g by setting the neutral keybits to random. Then, we compute the bias $|\varepsilon_g^\star|$ of the probability that the new function depending only on $256 - n$ keybits behaves as the full f function. Considering Aumasson's 36 PNBs, 100 executions, 2^5 keys and 2^{24} initial values, we obtained an average median bias of 0.005004, substantially higher than what was obtained in [AFK+08]. It is no surprise that their value may be pessimistic, given that another research reported a bias 4 times higher [MGM15]. Using the dual differential, we obtained the bias 0.004435. If we consider the conditioned scenario on x_7 and build the function g with our 37 BPNKBs, we obtain a bias $|\varepsilon_g^\star| = 0.004491$. When we extend conditions to the sixth word as well, we get $|\varepsilon_g^\star| = 0.004026$, using 38 BPNKBs. Needless to say, the chosen IVs for these experiments satisfy the condition that only 4 differences appear after the first Salsa round.

We performed further experiments regarding the threshold of the neutrality measure. Similarly to what was previously done in [MGM15], we set $\gamma = 0.04$ and found 48 BPNBs for Aumasson's differential. This configuration produces smaller biases, but in exchange, the parameter m is smaller. We observed a backward bias of 0.000265 and 0.000439 for both conditioned scenarios.

Forward PNBs

We came up with the notion of forward probabilistic neutral bits to name the initial bits that do not influence the intermediate state with high probability. In other words, let X, X' be two initial states satisfying the $\mathcal{ID}$ and $\overline{X}, \overline{X'}$ be these states with one bit flipped, then b_i is a FPNB if the bias of $(X^4_{\mathcal{OD}} \oplus X'^4_{\mathcal{OD}} \oplus \overline{X}^4_{\mathcal{OD}} \oplus \overline{X'}^4_{\mathcal{OD}} = 0 | \Delta^0_{\mathcal{ID}} = 1)$ is higher than a given threshold. Our hypothesis, with certain independency assumptions, is that fixing only the most significant bits of the IV (*i.e.* not FPNBs), will result in a higher forward bias.

Using Aumasson's differential and a threshold of 0.12, we detected 48 FPNBs in the IV (must not count the $\mathcal{ID}$). Similarly, using the differential $(\Delta^4_{6,14} | \Delta^0_{8,31})$, we detected 130 FPNBs, 43 of them in the IV. As in the case of BPNBs, these bits follow a pattern that can be observed in Table 11. Note the positions of the FPNBs are not random, but they appear in sequential order. Moreover, half of the MSB are FPNBs as well (at $a, b, d, \vec{a}, \vec{b}, \vec{c}, \vec{d}, \overline{c}$). When we consider the conditioned scenario on x_7 we obtain 5 additional IV FPNBs $x_{6,11}$, $x_{7,1}$, $x_{7,30}$, $x_{8,10}$, $x_{9,18}$. This number increases even more when we set conditions on x_6 as well, where we get 2 more IV FPNBs at $x_{7,20}$ and $x_{8,13}$.

We verified our hypothesis running Merengue to compute the forward bias using 2^{24} samples, sufficient to be detected. Freeing the FPNBs in the IV for sampling resulted in higher $|\varepsilon_f^\star|$ for all three cases; the standard, conditioned on x_7, conditioned also on x_6. The exact values were $0.235291, 0.255981, 0.271301$, respectively. Nonetheless, we obtained slightly lower values in the backward direction with $|\varepsilon_g^\star|$ equal to $0.003244, 0.003790, 0.001642$, each. Thus, this strategy will only be followed if the increase of ε_f is sufficient to outweight ε_g to obtain a total bias ε still higher than the case without FPNBs.

Setting a higher threshold to 0.2 decreases the number of FPNBs down to 11. In this case, the forward bias will be closer to the standard case, but the observed backward bias will grow instead. In particular, if no conditions are imposed we get $|\varepsilon_f^\star| = 0.154175$, we measure $|\varepsilon_f^\star| = 0.262024$ if we condition on $x_{7,12}$ and $|\varepsilon_f^\star| = 0.263672$ when we also flip $x_{6,10}$.

IV word	Intraword bit
x_6	0, 10, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31
x_7	1, 18, 19
x_8	1, 2, 3, 11, 12, 23, 24, 25, 26, 27, 28, 29, 30, 31
x_9	5, 6, 7, 8, 9, 10, 11, 13, 14, 15, 16, 17, 24, 25, 27, 28, 29, 30, 31

TABLE 5: 48 FPNBs of Aumasson's differential

9 Rounds

Today, there is no known key recovery attack on 9 rounds of Salsa (note [MGM15] gives a related-key attack). Despite our trials, we were unable to measure significant biases after 5 QuarterRound using single bit differentials. Unfortunately, this is no surprise in forward direction, as previously noted in [Ish12], nor in backward direction, see [AFK+08]. While a significant increase in the biases was observed, our strategies were not enough for now to achieve one more round. The next paragraphs give some insight on the exact ideas that we followed when trying to meet this goal, which we trust will be met anytime soon.

An important drawback of multibit differentials is the huge fall in the number of BPNBs, hence an increase in the time complexity of the attacker. Assuming single bit differentials, we still wanted to measure backward PNBs from the 9^{th} round for a better understanding of their distribution. The naive $\mathcal{OD} = \Delta^4_{1,14} | \mathcal{ID} = \Delta^0_{7,31}$ for $R = 9$ did not show any single BPNB. Conversely, $\Delta^5_{1,14}$ does not have any FPNB that could rise the forward bias. Nonetheless, the latter turned to have only one neutral bit with $\gamma_{14,31} = 0.308$.

If we look at the cipher structure, we can check $\Delta^4_{1,14}$ strongly affects the ninth word of the fifth round, since $x^5_{9,21} = x^4_{9,21} \oplus (x^4_{1,14} + x^4_{5,14}) \lll 7$. We posit most PNBs at $x^4_{1,14}$ will go to $x^5_{9,21}$, which is evidenced when computing its BPNBs: most BPNBs found at $x^4_{1,14}$ have their equivalent neutral bit at $x^5_{9,21}$, with similar neutrality measure (somewhat smaller, reason why there are fewer of them). In particular, we found 25 such BPNKBs with $\gamma = 0.12$, following a funny relation. Intuitively, intraword bit indexes must be rotated 7 bits left. Plus, these cycles hold for word indexes (from $(d, 14)^4$ to $(b, 21)^5$): $\vec{d} \mapsto \vec{b} \mapsto \overleftarrow{d} \mapsto \overleftarrow{b}$ and $\overline{\overline{d}} \mapsto \overline{\overline{b}} \mapsto d$, deriving from the transpose between consecutive rounds. The bad news is that there is no significant ε_f in $\Delta^5_{9,21}$, partly because we could not find any FPNB.

We tried to find BPNBs from the 9^{th} round at $x^5_{6,1}$, so as to get some estimates on a related key attack using the differential in [FMB+06]. Since this is a fourth order single bit differential, finding a number of BPNBs of a single $\mathcal{OD}$ did not seem impossible. Unfortunately, we only found one BPNB with $\gamma_{3,18} = 0.180$. Conversely, when we compute the BPNBs in the triple bit $\mathcal{OD} = \Delta^5_{9,0} \oplus \Delta^5_{13,0} \oplus \Delta^5_{1,13}$ in [CM17], we get none. If we compute them from the 8^{th} round, we get almost the same 36 Aumasson's PNBs, with indexes rotated one bit right. This makes total sense, because $x^4_{1,13} = x^5_{1,13} \oplus x^5_{9,0} \oplus x^5_{13,0}$.

4.3 Revisiting Existing Attacks

In the course of the preceding months, we have actively used our toolkit Merengue to support our arguments in practice. Along these lines, we verified the proper functioning of this software replicating Aumasson's attack [AFK+08] (among many others). In this case, we found almost twice the values ε_g and ε. However, we were unable to obtain similar results than those of Maitra [Mai16]. This section gives some insight into the observed differences.

As mentioned earlier, there exists a tradeoff between the threshold on the neutrality measure γ and the total bias ε. The former determines the number of BPNBs, and thus, the number of significant keybits m. The latter will affect the number of pairs N needed to perform the attack. Another important parameter is α, which must be chosen in a way that the first addend of the time complexity expression overcomes the second $2^m(\frac{\sqrt{\alpha \log 4}+3\sqrt{1-\varepsilon^2}}{\varepsilon})^2 + 2^{256-\alpha}$.

In his article, the author decided to include less BPNBs (corresponding to keywords in the fourth column) in order to observe a higher backward bias ε_g. Thanks to the exhaustive search performed over x_7 for each key, he could measure the bias of the only samples with suitable IVs. Using the same differential, he observed an improved forward bias with respect to Aumasson's of 0.228538. Applying our method of conditional cryptanalysis, we obtained the same bias with no need of expensive searches.

Our discrepancy comes next. Even if we also set the same 33 BPNBs, we have never got such a high backward bias. Maitra claimed a value $\varepsilon_g^\star = 0.013778 \approx 2^{-6}$ which, together with the forward bias, results in a total bias of $\varepsilon^\star = 0.003154 \approx 2^{-8}$. Nonetheless, we obtained a backward bias of $|\varepsilon_g^\star| = 0.020763$ and total bias $|\varepsilon^\star| = 0.002446$, meaning a difference of 1.29 times smaller. Needless to say, this mismatch has an impact on the overall attack complexity. While Maitra's published result was the fastest known attack so far with time complexity of $2^{245.52}$ and $2^{22.5}$ data, our reproduction of this analysis gives $2^{22.5}$ data and $2^{245.6}$ time using $\alpha = 15$. We attribute this difference to an insufficiently large sample leading to an unimportant miscalculation of the bias.

Anyhow, we do not understand how this analysis would translate into a real attack, where the adversary cannot access the number of differences after the first round. In the event of an actual secret key, the attacker cannot distinguish the IVs that give only 4 differences after the first QuarterRound. Instead, half of his samples would satisfy this constraint whereas the other half would not. Thus, this analysis is very good to estimate theoretical biases, though the author does not provide a concrete strategy to detect them in a real setting. On the contrary, our conditioned IV method would execute two threads and select the one with the higher bias observed (and thus, we must multiply our equivalent attack time complexity by 2).

Whilst we do not say that his result is false, we do think further details should be given that explain the exact strategy to be followed by an attacker. In the meantime, we intend to write the author to provide our views on this analysis.

5 Conclusion

This document gathers our study of the cipher Salsa, faster attacks to the version of 8 rounds and new tradeoffs. Our best attack has theoretical time complexity of $2^{241.8}$ and $2^{31.7}$ data. We found this complexity combining 4 conditioned threads on $x_{7,12}$ and $x_{6,10}$, 48 BPNBs with $\gamma = 0.04$ and 11 FPNBs. At the end of this internship, we provided plenty of new estimates on the cost of breaking 256-bit key Salsa20/8 as well as a technique of conditional cryptanalysis that could be further exploited in order to define tighter security margins for this family of ciphers. We have given some theoretical background on the choice of good differential pairs, based on its structure and a linearized version of Salsa, that answers to the pattern followed by most known differentials. We developed the toolkit Merengue, a useful set of functions that can be applied to cryptanalyse Salsa and other ARX ciphers.

Our next goal for the medium-term is to achieve the first attack to 9 rounds of Salsa. We will try to adapt our conditioned IVs strategy to move some significant forward bias in the 5^{th} round, using a reduced number of $\mathcal{OD}$. Regarding PNBs, we will study the differences (if any) when fixing significant IV bits and neutral keybits to zeroes, ones or random bits. Also, we will explore a new idea that aims at considering some concrete values of possibly weak keybits, so as to obtain a specific bitstring after undoing the feedforward. We want to check if there were any keybits of X^R whose value affects significantly any intermediate bit values.

Despite all the previous work done in this matter, no one has ever presented a key recovery attack to Salsa for over 8 rounds (which is not the case for related key attacks). However, we expect that these notions may imply a step forward in the security margin of this eSTREAM candidate.

We present here estimations of the attacker strategies we considered, showing the best possible combination of approaches. The reader may refer to the attached Table 6 to have a wider view of the parameters involved (*i.e.* number of rounds, input differential, output differential, conditioned initial values, forward bias, backward bias, total bias, backward PNBs of the key, forward PNBs of the initial value and negative exponent of the probability of false alarm base-2). Note that the rows with 11 FPNBs are not actual attacks because they leave fewer free IV bits than the number of pairs needed to detect the total bias. Nonetheless, they come in handy for theoretical reasons. The entries of the table are ordered from the slower attack to the fastest. The last two rows correspond to the most recent attacks to Salsa20/8; [Mai16] and [CM17], respectively, for the sake of comparison.

R	$\mathcal{ID}$	$\mathcal{OD}$	CIV	$\lvert\varepsilon_f^\star\rvert$	$\lvert\varepsilon_g^\star\rvert$	$\lvert\varepsilon^\star\rvert$	BPNB	FPNB	α	Data	Time
8	$\Delta^0_{8,31}$	$\Delta^4_{6,14}$		0.142761	0.004435	0.000543	30		7	$2^{26.4}$	$2^{252.5}$
8	$\Delta^0_{8,31}$	$\Delta^4_{6,14}$	x_8	0.264526	0.014184	0.001031	31		10	$2^{24.7}$	$2^{250.8}$
8	$\Delta^0_{7,31}$	$\Delta^4_{1,14}$	x_7	0.262024	0.017698	0.000890	33	11	12	$2^{25.3}$	$2^{249.4}$
8	$\Delta^0_{7,31}$	$\Delta^4_{1,14}$	x_7	0.262024	0.003081	0.000262	37	11	12	$2^{28.8}$	$2^{248.9}$
8	$\Delta^0_{7,31}$	$\Delta^4_{1,14}$	$x_7 \cup x_6$	0.263672	0.001418	0.000282	38	11	13	$2^{28.7}$	$2^{248.8}$
8	$\Delta^0_{7,31}$	$\Delta^4_{1,14}$	$x_7 \cup x_6$	0.252594	0.004026	0.000334	38		15	$2^{28.3}$	$2^{248.3}$
8	$\Delta^0_{7,31}$	$\Delta^4_{1,14}$	x_7	0.229462	0.004491	0.000368	37		12	$2^{27.8}$	$2^{248.0}$
8	$\Delta^0_{7,31}$	$\Delta^4_{1,14}$		0.154175	0.005649	0.000481	36	11	15	$2^{27.2}$	$2^{247.2}$
8	$\Delta^0_{7,31}$	$\Delta^4_{1,14}$		0.131409	0.005004	0.000585	36		12	$2^{26.5}$	$2^{246.7}$
8	$\Delta^0_{7,31}$	$\Delta^4_{1,14}$	x_7	0.229462	0.020763	0.002446	33		15	$2^{22.5}$	$2^{246.6}$
8	$\Delta^0_{7,31}$	$\Delta^4_{1,14}$	$x_7 \cup x_6$	0.271301	0.001642	0.000445	38	54	15	$2^{27.4}$	$2^{246.5}$
8	$\Delta^0_{7,31}$	$\Delta^4_{1,14}$		0.235291	0.003244	0.000763	36	47	15	$2^{25.9}$	$2^{245.9}$
8	$\Delta^0_{7,31}$	$\Delta^4_{1,14}$	x_7	0.255981	0.003790	0.000970	37	52	16	$2^{25.2}$	$2^{245.3}$
8	$\Delta^0_{7,31}$	$\Delta^4_{1,14}$	x_7	0.262024	0.000206	0.000054	48	11	19	$2^{33.7}$	$2^{242.7}$
8	$\Delta^0_{7,31}$	$\Delta^4_{1,14}$	x_7	0.255981	0.000212	0.000054	48	52	19	$2^{33.7}$	$2^{242.7}$
8	$\Delta^0_{7,31}$	$\Delta^4_{1,14}$	x_7	0.229462	0.000265	0.000061	48		19	$2^{33.4}$	$2^{242.4}$
8	$\Delta^0_{7,31}$	$\Delta^4_{1,14}$	$x_7 \cup x_6$	0.252594	0.000439	0.000111	48		21	$2^{31.7}$	$2^{241.8}$
8	$\mathbf{\Delta^0_{7,31}}$	$\mathbf{\Delta^4_{1,14}}$	$\boldsymbol{x_7 \cup x_6}$	**0.271301**	**0.000405**	**0.000110**	**48**	**54**	**17**	$\mathbf{2^{31.7}}$	$\mathbf{2^{241.8}}$
8	$\Delta^0_{7,31}$	$\Delta^4_{1,14}$	$x_7 \cup x_6$	0.263672	0.000446	0.000118	48	11	21	$2^{31.5}$	$2^{241.6}$
8	$\Delta^0_{7,31}$	$\Delta^4_{1,14}$	x_7	0.228538	0.013778	0.003154	33		15	$2^{22.5}$	$2^{245.5}$
8	$\Delta^0_{7,0}$	$\Delta^5_{9,0}$ $\Delta^5_{9,0}$ $\Delta^5_{13,0}$ $\Delta^5_{1,13}$	x_7	0.233198	0.000752	0.000178	42		15.5	$2^{30.8}$	$2^{244.9}$

TABLE 6: Results of our cryptanalysis of Salsa20/8

Appendix A. References

[AFK+08] Jean-Philippe Aumasson, Simon Fischer, Shahram Khazaei, Willi Meier, and Christian Rechberger. New Features of Latin Dances: Analysis of Salsa, ChaCha, and Rumba. In Kaisa Nyberg, editor, *Fast Software Encryption*, pages 470–488, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[BDK02] Eli Biham, Orr Dunkelman, and Nathan Keller. Enhancing differential-linear cryptanalysis. In Yuliang Zheng, editor, *Advances in Cryptology — ASIACRYPT 2002*, pages 254–266, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[Ber05] Daniel J. Bernstein. Salsa20. Technical Report 2005/025, eSTREAM, ECRYPT Stream Cipher Project, 2005. `http://cr.yp.to/snuffle.html`.

[Ber08a] Daniel J. Bernstein. Response to "Slid Pairs in Salsa20 and Trivium", 2008. `https://cr.yp.to/snuffle/reslid-20080925.pdf`.

[Ber08b] Daniel J. Bernstein. *The Salsa20 Family of Stream Ciphers*, pages 84–97. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[CM16] Arka Rai Choudhuri and Subhamoy Maitra. Differential Cryptanalysis of Salsa and ChaCha - An Evaluation with a Hybrid Model. *IACR Cryptology ePrint Archive*, 2016:377, 2016.

[CM17] Arka Choudhuri and Subhamoy Maitra. Significantly Improved Multi-bit Differentials for Reduced Round Salsa and Chacha. *IACR Transactions on Symmetric Cryptology*, 2016(2):261–287, 2017.

[Cro06] Paul Crowley. Truncated differential cryptanalysis of five rounds of salsa20. eSTREAM, ECRYPT Stream Cipher Project, Report 2006/010, 2006. `http://www.ecrypt.eu.org/stream`.

[FMB+06] Simon Fischer, Willi Meier, Côme Berbain, Jean-François Biasse, and M. J. B. Robshaw. Non-randomness in eSTREAM Candidates Salsa20 and TSC-4. In Rana Barua and Tanja Lange, editors, *Progress in Cryptology - INDOCRYPT 2006*, pages 2–16, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[Ish12] Tsukasa Ishiguro. Modified version of "Latin Dances Revisited: New Analytic Results of Salsa20 and ChaCha". Cryptology ePrint Archive, Report 2012/065, 2012. `https://eprint.iacr.org/2012/065`.

[KMNP10] Simon Knellwolf, Willi Meier, and María Naya-Plasencia. Conditional Differential Cryptanalysis of NLFSR-Based Cryptosystems. In Masayuki Abe, editor, *Advances in Cryptology - ASIACRYPT 2010*, pages 130–145, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[KN10] Dmitry Khovratovich and Ivica Nikolić. Rotational Cryptanalysis of ARX. In Seokhie Hong and Tetsu Iwata, editors, *Fast Software Encryption*, pages 333–346, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[LM02] Helger Lipmaa and Shiho Moriai. Efficient Algorithms for Computing Differential Properties of Addition. In Mitsuru Matsui, editor, *Fast Software Encryption*, pages 336–350, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[Mai16] Subhamoy Maitra. Chosen IV Cryptanalysis on Reduced Round ChaCha and Salsa. *Discrete Appl. Math.*, 208(C):88–97, July 2016.

[Mat94] Mitsuru Matsui. Linear cryptanalysis method for des cipher. In Tor Helleseth, editor, *Advances in Cryptology — EUROCRYPT '93*, pages 386–397, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.

[MGM15] Subhamoy Maitra, Paul Goutam, and Willi Meier. Salsa20 Cryptanalysis: New Moves and Revisiting Old Styles. In Jean-Pierre Tillich Pascale Charpin, Nicolas Sendrier, editor, *The 9th International Workshop on Coding and Cryptography 2015 WCC2015*, Proceedings of the 9th International Workshop on Coding and Cryptography 2015 WCC2015, Paris, France, April 2015. Anne Canteaut, Gaëtan Leurent, Maria Naya-Plasencia.

[MS02] Itsik Mantin and Adi Shamir. A practical attack on broadcast rc4. In Mitsuru Matsui, editor, *Fast Software Encryption*, pages 152–164, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[PSB08] Deike Priemuth-Schmid and Alex Biryukov. Slid Pairs in Salsa20 and Trivium. In Dipanwita Roy Chowdhury, Vincent Rijmen, and Abhijit Das, editors, *Progress in Cryptology - INDOCRYPT 2008*, pages 1–14, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[Sie85] Thomas Siegenthaler. Decrypting a Class of Stream Ciphers Using Ciphertext Only. *IEEE Transactions on Computers*, C-34:81 – 85, 02 1985.

[SZFW13] Zhenqing Shi, Bin Zhang, Dengguo Feng, and Wenling Wu. Improved Key Recovery Attacks on Reduced-Round Salsa20 and ChaCha. In Taekyoung Kwon, Mun-Kyu Lee, and Daesung Kwon, editors, *Information Security and Cryptology – ICISC 2012*, pages 337–351, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[TSK+07] Yukiyasu Tsunoo, Teruo Saito, Hiroyasu Kubo, Tomoyasu Suzaki, and Hiroki Nakashima. Differential Cryptanalysis of Salsa20/8. eSTREAM, ECRYPT Stream Cipher Project, Report 2007/010, 2007. `http://www.ecrypt.eu.org/stream`.

[VMDCP11] Vesselin Velichkov, Nicky Mouha, Christophe De Cannière, and Bart Preneel. The Additive Differential Probability of ARX. In *Proceedings of the 18th International Conference on Fast Software Encryption*, FSE'11, pages 342–358, Berlin, Heidelberg, 2011. Springer-Verlag.

Appendix B. Tables and Figures

Word	Common NBs	Except	Extra NBs
(c, j)	$(\vec{c}, 18 + j)$	$(c, 12)$	$(c, 31)$
		$(c, 25)$	$(c, 31)$
		$(c, 26)$	$(\vec{c}, 13)$
(d, j)	$(\vec{c}, 28 + j)$	$(d, 7)$	$(\vec{c}, 13), (\vec{c}, 22), (\vec{b}, 13), (\vec{b}, 26)$
	$(\bar{\vec{c}}, 8 + j)$	$(d, 9)$	$(\bar{\vec{c}}, 31)$
	$(\bar{\vec{c}}, 20 + j)$	$(d, 12)$	$(\vec{b}, 31)$
	$(\bar{\vec{c}}, 21 + j)$	$(d, 14)$	$(\bar{\vec{c}}, 4)$
	$(\bar{\vec{c}}, 27 + j)$	$(d, 16)$	$(\vec{c}, 31)$
		$(d, 17)$	$(\bar{\vec{c}}, 13)$
		$(d, 22)$	$(\bar{\vec{c}}, 31)$
		$(d, 25)$	$(\vec{b}, 31)$

TABLE 7: Neutral bits after 4 rounds of linearized Salsa

ADD	Δ_{out}	Δ_{carry}				
$\mathbf{(0+0)^0}$	**0**	**0**	$(0+0)^0/(0+0)^0 = 0^0/0^0$	$(0+1)^0/(0+1)^0 = 1^0/1^0$	$(1+0)^0/(1+0)^0 = 1^0/1^0$	$(1+1)^0/(1+1)^0 = 0^1/0^1$
	0	**0**	$(0+0)^1/(0+0)^1 = 1^0/1^0$	$(0+1)^1/(0+1)^1 = 0^1/0^1$	$(1+0)^1/(1+0)^1 = 0^1/0^1$	$(1+1)^1/(1+1)^1 = 1^1/1^1$
$\mathbf{(0+0)^1}$	**1**	**0.5**	$(0+0)^0/(0+0)^1 = 0^0/1^0$	$(0+1)^0/(0+1)^1 = 1^0/0^1$	$(1+0)^0/(1+0)^1 = 1^0/0^1$	$(1+1)^0/(1+1)^1 = 0^1/1^1$
	1	**0.5**	$(0+0)^1/(0+0)^0 = 1^0/0^0$	$(0+1)^1/(0+1)^0 = 0^1/1^0$	$(1+0)^1/(1+0)^0 = 0^1/1^0$	$(1+1)^1/(1+1)^0 = 1^1/0^1$
$\mathbf{(0+1)^0}$	**1**	**0.5**	$(0+0)^0/(0+1)^0 = 0^0/1^0$	$(0+1)^0/(0+0)^0 = 1^0/0^0$	$(1+0)^0/(1+1)^0 = 1^0/0^1$	$(1+1)^0/(1+0)^0 = 0^1/1^0$
	1	**0.5**	$(0+0)^1/(0+1)^1 = 1^0/0^1$	$(0+1)^1/(0+0)^1 = 0^1/1^0$	$(1+0)^1/(1+1)^1 = 0^1/1^1$	$(1+1)^1/(1+0)^1 = 1^1/0^1$
$\mathbf{(0+1)^1}$	**0**	**0.5**	$(0+0)^0/(0+1)^1 = 0^0/0^1$	$(0+1)^0/(0+0)^1 = 1^0/1^0$	$(1+0)^0/(1+1)^1 = 1^0/1^1$	$(1+1)^0/(1+0)^1 = 0^1/0^1$
	0	**0.5**	$(0+0)^1/(0+1)^0 = 1^0/0^0$	$(0+1)^1/(0+0)^0 = 0^1/0^0$	$(1+0)^1/(1+1)^0 = 0^1/0^1$	$(1+1)^1/(1+0)^0 = 1^1/1^0$
$\mathbf{(1+0)^0}$	**1**	**0.5**	$(0+0)^0/(1+0)^0 = 0^0/1^0$	$(0+1)^0/(1+1)^0 = 1^0/0^1$	$(1+0)^0/(0+0)^0 = 1^0/0^0$	$(1+1)^0/(0+1)^0 = 0^1/1^0$
	1	**0.5**	$(0+0)^1/(1+0)^1 = 1^0/0^1$	$(0+1)^1/(1+1)^1 = 0^1/1^1$	$(1+0)^1/(0+0)^1 = 0^1/1^0$	$(1+1)^1/(0+1)^1 = 1^1/0^1$
$\mathbf{(1+0)^1}$	**0**	**0.5**	$(0+0)^0/(1+0)^1 = 0^0/0^1$	$(0+1)^0/(1+1)^1 = 1^0/1^1$	$(1+0)^0/(0+0)^1 = 1^0/1^0$	$(1+1)^0/(0+1)^1 = 0^1/0^1$
	0	**0.5**	$(0+0)^1/(1+0)^0 = 1^0/1^0$	$(0+1)^1/(1+1)^0 = 0^1/0^1$	$(1+0)^1/(0+0)^0 = 0^1/0^0$	$(1+1)^1/(0+1)^0 = 1^1/1^0$
$\mathbf{(1+1)^0}$	**0**	**0.5**	$(0+0)^0/(1+1)^0 = 0^0/0^1$	$(0+1)^0/(1+0)^0 = 1^0/1^0$	$(1+0)^0/(0+1)^0 = 1^0/1^0$	$(1+1)^0/(0+0)^0 = 0^1/0^0$
	0	**0.5**	$(0+0)^1/(1+1)^1 = 1^0/1^1$	$(0+1)^1/(1+0)^1 = 0^1/0^1$	$(1+0)^1/(0+1)^1 = 0^1/0^1$	$(1+1)^1/(0+0)^1 = 1^1/1^0$
$\mathbf{(1+1)^1}$	**1**	**1**	$(0+0)^0/(1+1)^1 = 0^0/1^1$	$(0+1)^0/(1+0)^1 = 1^0/0^1$	$(1+0)^0/(0+1)^1 = 1^0/0^1$	$(1+1)^0/(0+0)^1 = 0^1/1^0$
	1	**1**	$(0+0)^1/(1+1)^0 = 1^0/0^1$	$(0+1)^1/(1+0)^0 = 0^1/1^0$	$(1+0)^1/(0+1)^0 = 0^1/1^0$	$(1+1)^1/(0+0)^0 = 1^1/0^0$

TABLE 8: Differential Addition

CIV	d_{31}	d_{26}	d_{25}	d_{24}	d_{23}	d_{18}	d_{17}	d_{14}
NULL		**0.49999**						**0.48675**
$(\bar{\bar{c}}, 12)$		**0.49963**		**0.50053**	**0.49997**			**0.45543**
$(6, 10)$	**0.49995**	**0.52063**	**0.49447**	**0.49937**	**0.50001**	**0.49837**	**0.49988**	**0.54782**
d_{13}	d_{12}	d_{11}	d_{10}	d_9	d_7	d_5	d_4	d_1
	0.50046	**0.49996**						
0.49738	**0.50122**	**0.49987**	**0.49999**			**0.50004**		**0.49941**
	0.49986			**0.50004**	**0.50131**	**0.49549**	**0.49999**	
d_0	$\vec{d}_{26}$	$\vec{d}_{25}$	$\vec{d}_{24}$	$\vec{d}_{19}$	$\vec{d}_{17}$	$\vec{d}_{16}$	$\vec{d}_7$	
	0.49970	**0.49994**			**0.50003**			
0.50002	**0.49749**	**0.49930**	**0.50002**		**0.50041**	**0.49995**		
0.49915				**0.50001**	**0.50018**	**0.49999**	**0.49986**	

TABLE 9: PDARX using conditioned x_7

$\mathcal{ID}$ $(\bar{\bar{c}}, 31)$	$\mathcal{OD}$ $(d, 14)$	BPNBs 63	64
Word	Bits	$\gamma_i > 0.12$	
d	26	0.734	0.737
	27	0.622	0.625
	28	0.490	0.493
	29	0.355	0.359
	30	0.234	0.239
	31	0.153	0.158
$\vec{a}$	21	0.142	0.147
	25	0.248	0.252
	26	0.123	0.128
	31	0.179	0.184
$\overleftarrow{b}$	24	0.370	0.373
	25	0.245	0.250
	26	0.141	0.145
$\vec{d}$	13	0.261	0.265
	14	0.141	0.145
	31	0.286	0.290
c	18	0.606	0.609
	19	0.411	0.415
	20	0.243	0.247

$\bar{\bar{a}}$	4	0.779	0.781
	5	0.676	0.680
	6	0.548	0.551
	7	0.401	0.405
	8	0.255	0.258
	9	0.131	0.135
	18	0.674	0.677
	19	0.502	0.506
	20	0.299	0.303
	24	0.123	0.126
$\vec{b}$	0	0.447	0.451
	1	0.269	0.273
	2		0.122
	18	0.779	0.782
	19	0.676	0.679
	20	0.548	0.551
	21	0.401	0.405
	22	0.255	0.259
	23	0.130	0.134
$\bar{\bar{b}}$	7	0.337	0.341
	8	0.197	0.200
$\bar{\bar{d}}$	20	0.129	0.131

$\overleftarrow{d}$	5	0.998	1.000
	6	0.996	0.998
	7	0.993	0.994
	8	0.987	0.990
	9	0.977	0.980
	10	0.959	0.963
	11	0.929	0.936
	12	0.880	0.889
	13	0.804	0.814
	14	0.691	0.703
	15	0.547	0.563
	16	0.376	0.393
	17	0.201	0.219
$\overleftarrow{a}$	22	0.784	0.786
	23	0.674	0.677
	24	0.537	0.540
	25	0.371	0.376
	26	0.230	0.234
a	10	0.387	0.386
	11	0.263	0.268
	12	0.155	0.160
	30	0.468	0.471
	31	0.373	0.377

TABLE 10: Backward Probabilistic Neutral Bits on $8-4$ Salsa reverse rounds

$\mathcal{ID}$ $(\bar{\bar{c}}, 31)$	$\mathcal{OD}$ $(d, 14)$	FPNBs 134	130
Word	Bits	$d = 1$	$d = 6$
$\vec{c}$	0	0.184	0.184
	1	0.158	0.159
	2	0.130	0.129
	7	0.219	0.220
	8	0.213	0.215
	9	0.226	0.224
	10	0.224	0.223
	11	0.224	0.223
	12	0.225	0.224
	13	0.218	0.218
	14	0.212	0.209
	15	0.231	0.228
	16	0.220	0.219
	17	0.224	0.223
	18	0.228	0.222
	19	0.162	0.162
	20	0.129	0.131
	25	0.228	0.227
	26	0.208	0.210
	27	0.187	0.196
	28	0.248	0.232
	29	0.225	0.219
	30	0.215	0.217
	31	0.223	0.223
d	0	0.198	0.197
	1	0.161	0.154
	2	0.159	0.153
	3	0.143	0.140
	4	0.121	
	6	0.190	0.176
	7	0.162	0.171
	8	0.156	0.143
	9	0.149	0.125
	17	0.131	0.129
	18	0.128	0.125
	20	0.193	0.183
	21	0.159	0.153
	22	0.151	0.146
	23	0.127	
	30	0.210	0.213
	31	0.208	0.206
$\overleftarrow{b}$	24	0.149	0.135
	25	0.126	0.130
c	1	0.145	0.143
	2	0.131	0.129
	4	0.210	0.212
	5	0.197	0.195
	6	0.194	0.195
	7	0.170	0.170
	8	0.144	0.142
	14	0.225	0.226
	15	0.225	0.225
	16	0.223	0.224
	17	0.223	0.224
	18	0.224	0.225
	19	0.223	0.222
	20	0.222	0.221
	21	0.220	0.220
	22	0.215	0.215
	23	0.208	0.207
	24	0.194	0.193
	25	0.177	0.176
	26	0.154	0.152
	27	0.127	0.127
$\vec{d}$	0	0.140	0.138
	10	0.129	0.149
	22	0.135	0.133
	23	0.218	0.219
	24	0.217	0.216
	25	0.212	0.216
	26	0.213	0.209
	27	0.201	0.205
	28	0.201	0.198
	29	0.193	0.188
	30	0.184	0.183
	31	0.168	0.175
a	6	0.188	0.185
	7	0.176	0.169
	8	0.158	0.155
	9	0.146	0.133
	10	0.130	
	30	0.157	0.155
	31	0.141	0.141
$\vec{a}$	0	0.123	0.129
	30	0.175	0.179
	31	0.165	0.173
$\bar{\bar{c}}$	18	0.235	0.223
	19	0.169	0.158
b	5	0.215	0.215
	6	0.209	0.209
	7	0.199	0.198
	8	0.185	0.184
	9	0.169	0.169
	10	0.146	0.145
	11	0.126	0.126
	13	0.192	0.199
	14	0.189	0.180
	15	0.160	0.176
	16	0.154	0.161
	17	0.140	
	24	0.132	0.131
	25	0.130	0.128
	27	0.196	0.186
	28	0.173	0.182
	29	0.167	0.172
	30	0.140	0.154
	31	0.134	0.133
$\overleftarrow{c}$	1	0.167	0.169
	2	0.144	0.146
	3	0.121	0.121
	11	0.158	0.156
	12	0.136	0.130
	23	0.123	
	24	0.209	0.205
	25	0.203	0.194
	26	0.186	0.173
	27	0.163	0.162
	28	0.141	0.141
	29	0.135	0.135
	30	0.125	0.124
	31	0.126	0.125
$\vec{b}$	0	0.221	0.220
	1	0.216	0.219
	2	0.215	0.212
	3	0.212	0.210
	4	0.200	0.202
	5	0.199	0.202
	6	0.171	0.179
	7	0.152	0.145
	8		0.123
	17	0.155	0.154
	18	0.124	0.122
	29	0.136	0.135
	30	0.224	0.223
	31	0.231	0.229

TABLE 11: Forward Probabilistic Neutral Bits on 4 Salsa rounds

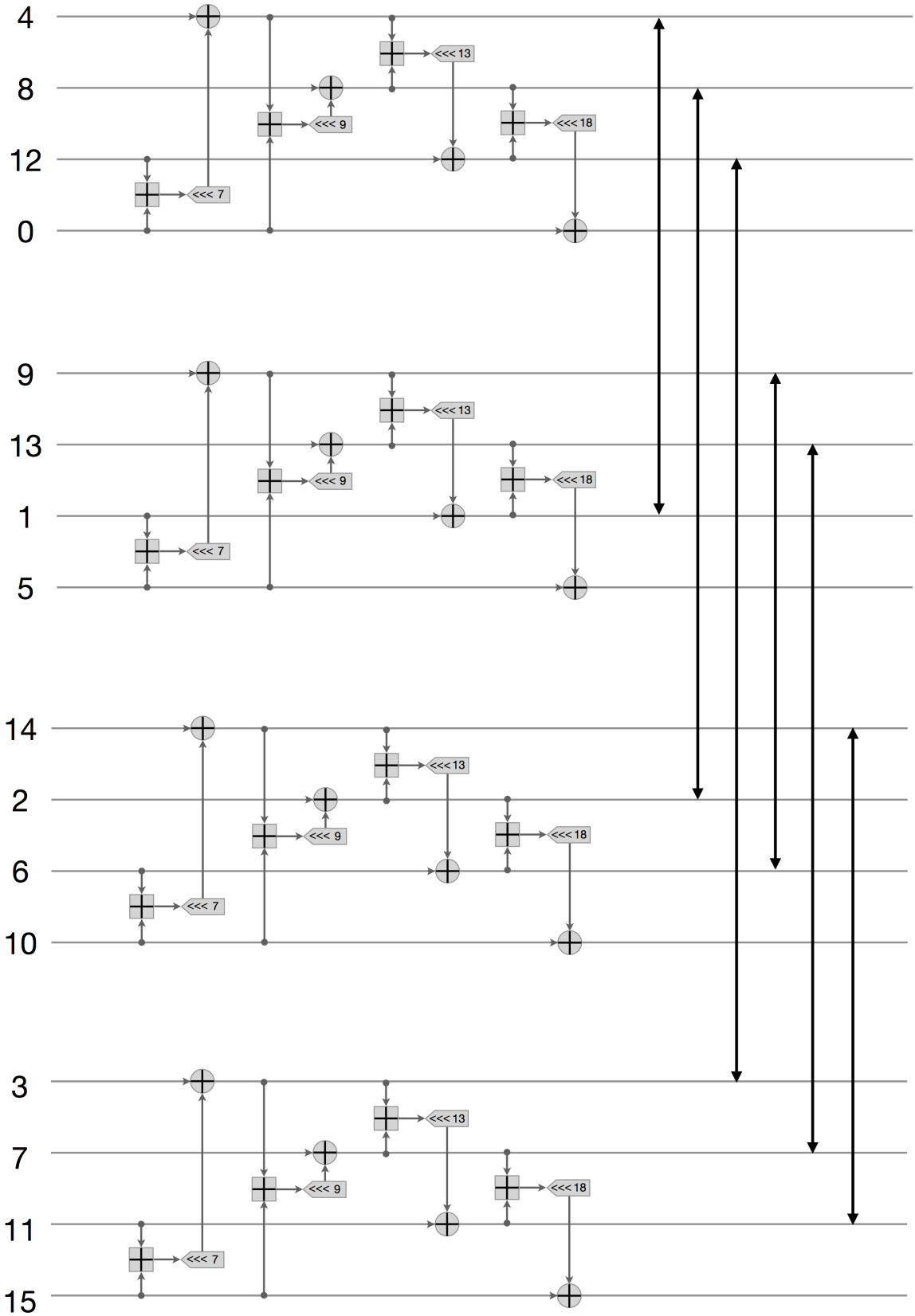


FIGURE 7: One QuarterRound function followed by a transposition