Efficient Algorithms for Computing Differential Properties of Addition

Helger Lipmaa[1] and Shiho Moriai[2]

[1] Helsinki University of Technology, Department of Computer Science and Engineering
P.O.Box 5400, FI-02015 HUT, Espoo, Finland
helger@tml.hut.fi
[2] NTT Laboratories
1-1 Hikari-no-oka, Yokosuka, 239-0847 Japan
shiho@isl.ntt.co.jp

Abstract. In this paper we systematically study the differential properties of addition modulo 2^n. We derive $\Theta(\log n)$-time algorithms for most of the properties, including differential probability of addition. We also present log-time algorithms for finding good differentials. Despite the apparent simplicity of modular addition, the best known algorithms require naive exhaustive computation. Our results represent a significant improvement over them. In the most extreme case, we present a complexity reduction from $\Omega(2^{4n})$ to $\Theta(\log n)$.

Keywords: modular addition, differential cryptanalysis, differential probability, impossible differentials, maximum differential probability.

1 Introduction

One of the most successful and influential attacks against block ciphers is Differential Cryptanalysis (DC), introduced by Biham and Shamir in 1991 [BS91a]. For many of the block ciphers proposed since then, provable security against DC (defined by Lai, Massey and Murphy [LMM91] and first implemented by Nyberg and Knudsen [NK95]) has been one of the primary criteria used to confirm their potential quality.

Unfortunately, few approaches to proving security have been really successful. The original approach of [NK95] has been used in designing MISTY and its variant KASUMI (the new 3GPP block cipher standard). Another influential approach has been the "wide trail" strategy proposed by Daemen [Dae95], applied for example in the proposed AES, Rijndael. The main reason for the small number of successful strategies is the complex structure of modern ciphers, which makes exact evaluation of their differential properties infeasible. This has, unfortunately, led to a situation where the security against DC is often evaluated by heuristic methods.

We approach the above problem by using the bottom-up methodology. That is, we evaluate many sophisticated differential properties of one of the most-used "non-trivial" block cipher cornerstones, *addition modulo* 2^n for $n \geq 1$. We hope that this will help to evaluate the differential properties of larger composite cipher parts like the Pseudo-Hadamard Transform, with the entire cipher being the final goal. The algorithms proposed here will enable the advanced cryptanalysis of block ciphers. We hope that our

results will facilitate cryptanalysis of such stream ciphers and hash functions that use addition and XOR at the same time.

Importance of Differential Properties of Addition. Originally, DC was considered with respect to XOR, and was generalized to DC with respect to an arbitrary group operation in [LMM91]. In 1992, Berson [Ber92] observed that for many primitive operations, it is significantly more difficult to apply DC with respect to XOR than with respect to addition modulo 2^{32}. Most interestingly, he classified DC of addition modulo 2^n itself, with n sufficiently big, with respect to XOR to be hard to analyze, given the (then) current state of theory.

Until now it has seemed that the problem of evaluating the differential properties of addition with respect to XOR *is* hard. Hereafter, we omit the "with respect to XOR" and take the addition to be always modulo 2^n. The fastest known algorithms for computing the differential probability of addition $\mathrm{DP}^{+}(\alpha, \beta \mapsto \gamma) := \mathbf{P}_{x,y}[(x+y) \oplus ((x \oplus \alpha) + (y \oplus \beta)) = \gamma]$ is exponential in n. The complexity of the algorithms for the maximum differential probability $\mathrm{DP}^{+}_{\mathsf{max}}(\alpha, \beta) := \max_{\gamma} \mathrm{DP}^{+}(\alpha, \beta \mapsto \gamma)$, the double-maximum differential probability $\mathrm{DP}^{+}_{\mathsf{2max}}(\alpha) := \max_{\beta,\gamma} \mathrm{DP}^{+}(\alpha, \beta \mapsto \gamma)$, and many other differential properties of addition are also exponential in n.

With small n (e.g., $n = 8$ or even with $n = 16$), exponential-in-n computation is feasible, as demonstrated in the cryptanalysis of FEAL by Aoki, Kobayashi and Moriai in [AKM98]. However, this is not the case when $n \geq 32$ as used in the recent 128-bit block ciphers such as MARS, RC6 and Twofish. In practice, if $n \geq 32$, both cipher designers and cryptanalysts have mostly made use of only a few differential properties of addition. (For example, letting x_0 be the least significant bit of x, they often use the property that $\alpha_0 \oplus \beta_0 \oplus \gamma_0 = 0$.) It means that block ciphers that employ both XOR and addition modulo 2^n are hard to evaluate the security against DC due to the lack of theory. This has led to the general concern that mixed use of XOR and modular addition might add more confusion (in Shannon's sense) to a cipher but "none has yet demonstrated to have a clear understanding of how to produce any proof nor convincing arguments of the advantage of such an approach" [Knu99]. One could say that they also add more confusion to the cipher in the layman's sense.

There has been significant ongoing work on evaluating the security of such "confusing" block ciphers against differential attacks. Some of these papers have also somewhat focused on the specific problem of evaluating the differential properties of addition. The full version of [BS91b] treated some differential probabilities of addition modulo 2^n and included a few formulas useful to compute DP^{+}, but did not include any concrete algorithms nor estimations of their complexities. The same is true for many later papers that analyzed ciphers like RC5, SAFER, and IDEA. Miyano [Miy98] studied the simpler case with one addend fixed and derived a linear-time algorithm for computing the corresponding differential probability.

Our Results. We develop a framework that allows the extremely efficient evaluation of many interesting differential properties of modular addition. In particular, most of the algorithms described herein run in time, sublinear in n. Since this would be impossible in the Turing machine model, we chose to use a realistic unit-cost RAM (*Random*

Access Machine) model, which executes basic n-bit operations like Boolean operations and addition modulo 2^n in unit time, as almost all contemporary microprocessors do.

The choice of this model is clearly motivated by the popularity of such microprocessors. Still, for several problems (although sometimes implicitly) we also describe linear-time algorithms that might run faster in hardware. (Moreover, the linear-time algorithms are usually easier to understand and hence serve an educational purpose.) Nevertheless, the RAM model was chosen to be "minimal", such that the described algorithms would be directly usable on as many platforms as possible. On the other hand, we immediately demonstrate the power of this model by describing some useful log-time algorithms (namely, for the Hamming weight, all-one parity and common alternation parity). They become very useful later when we investigate other differential properties. One of them (for the common alternation parity) might be interesting by itself; we have not met this algorithm in the literature.

After describing the model and the necessary tools, we show that DP^+ can be computed in time $\Theta(\log n)$ in the worst-case. The corresponding algorithm has two principal steps. The first step checks in constant-time whether the differential $\delta = (\alpha, \beta \mapsto \gamma)$ is impossible (i.e., whether $\mathrm{DP}^+(\delta) = 0$). The second step, executed only if δ is possible, computes the Hamming weight of an n-bit string in time $\Theta(\log n)$. As a corollary, we prove an open conjecture from [AKM98].

The structure of the described algorithm raises an immediate question of what is the density of the possible differentials. We show that the event $\mathrm{DP}^+(\delta) \neq 0$ occurs with the negligible probability $\frac{1}{2} \cdot \left(\frac{7}{8}\right)^{n-1}$ (This proves an open conjecture stated in [AKM98]). That is, the density of possible differentials is negligible, so DP^+ can be computed in time $\Theta(1)$ in the average-case. These results can be further used for impossible differential cryptanalysis, since the best previously known general algorithm to find non-trivial impossible differentials was by exhaustive search. Moreover, the high density of impossible differentials makes differential cryptanalysis more efficient; most of the wrong pairs can be filtered out [BS91a,O'C95].

Furthermore, we compute the explicit probabilities $\mathbf{P}_\delta[\mathrm{DP}^+(\delta) = i]$ for any $i,\ 0 \leq i \leq 1$. This helps us to compute the distribution of the random variable $X : \delta \mapsto \mathrm{DP}^+(\delta)$, and to create formulas for the expected value and variance of the random variable X. Based on this knowledge, one can easily compute the probabilities that $\mathbf{P}[X > i]$ for any i.

For the practical success of differential attacks it is not always sufficient to pick a random differential hoping it will be "good" with reasonable probability. It would be nice to find good differentials efficiently in a deterministic way. Both cipher designers and cryptanalysts are especially interested in finding the "optimal" differentials that result in the maximum differential probabilities and therefore in the best possible attacks. For this purpose we describe a log-time algorithm for computing $\mathrm{DP}^+_{\mathsf{max}}(\alpha, \beta)$ and a γ that achieves this probability. Both the structure of the algorithm (which makes use of the all-one parity) and its proof of correctness are nontrivial. We also describe a log-time algorithm that finds a pair (β, γ) that maximizes the double-maximum differential probability $\mathrm{DP}^+_{2\mathsf{max}}(\alpha)$. We show that for many nonzero α-s, $\mathrm{DP}^+_{2\mathsf{max}}(\alpha)$ is very close to one. A summary of some of our results is presented in Table 1.

	DP^+	$\mathrm{DP}^+_{\mathsf{max}}$	$\mathrm{DP}^+_{\mathsf{2max}}$
Previous result	$\Omega\left(2^{2n}\right)$	$\Omega\left(2^{3n}\right)$	$\Omega\left(2^{4n}\right)$
Our result	$\Theta(\log n)$ (worst-case), $\Theta(1)$ (average)	$\Theta(\log n)$	$\Theta(\log n)$

Table 1. Summary of the efficiency of our main algorithms

Road map. We give some preliminaries in Sect. 2. Section 3 describes a unit-cost RAM model, and introduces the reader to several efficient algorithms that are crucial for the later sections. In Sect. 4 we describe a log-time algorithm for DP^+. Section 5 gives formulas for the density of impossible differentials and other statistical properties of DP^+. Algorithms for maximum differential probability and related problems are described in Sect. 6.

2 Preliminaries

Let $\Sigma = \{0,1\}$ be the binary alphabet. For any n-bit string $x \in \Sigma^n$, let $x_i \in \Sigma$ be the i-th coordinate of x (i.e., $x = \sum_{i=0}^{n-1} x_i 2^i$). We always assume that $x_i = 0$ if $i \notin [0, n-1]$. (That is, $x = \sum_{-\infty}^{\infty} x_i 2^i$.)

Let $\oplus$, $\vee$, $\wedge$ and $\neg$ denote n-bit bitwise "XOR", "OR", "AND" and "negation", respectively. Let $x \gg i$ (resp. $x \ll i$) denote the right (resp. the left) shift by i positions (i.e., $x \gg i := \lfloor x/2^i \rfloor$ and $x \ll i := 2^i x \mod 2^n$). Addition is always performed modulo 2^n, if not stated otherwise. For any x, y and z we define $\mathsf{eq}(x,y,z) := (\neg x \oplus y) \wedge (\neg x \oplus z)$ (i.e., $\mathsf{eq}(x,y,z)_i = 1 \iff x_i = y_i = z_i$) and $\mathsf{xor}(x,y,z) := x \oplus y \oplus z$. For any n, let $\mathsf{mask}(n) := 2^n - 1$. For example, $((\neg 0) \ll 1)_0 = 0$.

Addition modulo 2^n. The *carry*, $\mathsf{carry}(x,y) := c \in \Sigma^n$, $x, y \in \Sigma^n$, of addition $x+y$ is defined recursively as follows. First, $c_0 := 0$. Second, $c_{i+1} := (x_i \wedge y_i) \oplus (x_i \wedge c_i) \oplus (y_i \wedge c_i)$, for every $i \geq 0$. Equivalently, $c_{i+1} = 1 \iff x_i + y_i + c_i \geq 2$. (That is, the carry bit c_{i+1} is a function of the *sum* $x_i + y_i + c_i$.) The following is a basic property of addition modulo 2^n.

Property 1. If $(x,y) \in \Sigma^n \times \Sigma^n$, then $x + y = x \oplus y \oplus \mathsf{carry}(x,y)$.

Differential Probability of Addition. We define the *differential* of addition modulo 2^n as a triplet of two input and one output differences, denoted as $(\alpha, \beta \mapsto \gamma)$, where $\alpha, \beta, \gamma \in \Sigma^n$. The *differential probability of addition* is defined as follows:

$$\mathrm{DP}^+(\delta) = \mathrm{DP}^+(\alpha, \beta \mapsto \gamma) := \mathbf{P}_{x,y}[(x+y) \oplus ((x \oplus \alpha) + (y \oplus \beta)) = \gamma] \ .$$

That is, $\mathrm{DP}^+(\delta) := \sharp\{x, y : (x+y) \oplus ((x \oplus \alpha) + (y \oplus \beta)) = \gamma\}/2^{2n}$. We say that δ is *impossible* if $\mathrm{DP}^+(\delta) = 0$. Otherwise we say that δ is *possible*. It follows directly from Property 1 that one can rewrite the definition of DP^+ as follows:

Lemma 1. $\mathrm{DP}^+(\alpha, \beta \mapsto \gamma) = \mathbf{P}_{x,y}[\mathsf{carry}(x,y) \oplus \mathsf{carry}(x \oplus \alpha, y \oplus \beta) = \mathsf{xor}(\alpha, \beta, \gamma)]$.

Probability Theory. Let X be a discrete random variable. Except for a few explicitly mentioned cases, we always deal with uniformly distributed variables. We note that in the *binomial distribution*, $\mathbf{P}[X = k] = p^k(1-p)^{n-k}\binom{n}{k} =: b(k; n, p)$, for some fixed $0 \leq p \leq 1$ and any $k \in \mathbb{Z}_{n+1}$. From the basic axioms of probability, $\sum_{k=0}^{n} b(k; n, p) = 1$. Moreover, the *expectation* $\mathbf{E}[X] = \sum_{k=0}^{n} k \cdot \mathbf{P}[X = k]$ of a binomially distributed random variable X is equal to np, while the *variance* $\mathbf{Var}[X] = \mathbf{E}[X^2] - \mathbf{E}[X]^2$ is equal to $np(1-p)$.

3 RAM Model and Some Useful Algorithms

In the n-bit unit-cost RAM model, some subset of fixed n-bit operations can be executed in constant time. In the current paper, we specify this subset to be a small set of n-bit instructions, all of which are readily available in the vast majority of contemporary microprocessors: Boolean operations, addition, and the constant shifts. We additionally allow unit-cost equality tests and (conditional) jumps. On the other hand, our model does *not* include table look-ups or (say) multiplications. Such a restriction guarantees that algorithms efficient in this model are also efficient on a very broad class of platforms, including FPGA and other hardware. This is further emphasized by the fact that our algorithms need only a few bytes of extra memory and thus a very small circuit size in hardware implementations.

Many algorithms that we derive in the current paper make heavy use of the three non-trivial functions described below. The power of our minimal computational model is stressed by the fact that all three functions can be computed in time $\Theta(\log n)$.

Hamming Weight. The first function is the *Hamming weight function* (also known as the *population count* or, sometimes, as *sideways addition*) $\mathsf{w_h}$: For $x = \sum_{i=0}^{n-1} x_i 2^i$, $\mathsf{w_h}(x) = \sum_{i=0}^{n-1} x_i$, i.e., $\mathsf{w_h}$ counts the "one" bits in an n-bit string. In the unit-cost RAM model, $\mathsf{w_h}(x)$ can be computed in $\Theta(\log n)$ steps. Many textbooks contain (a variation of) the next algorithm that we list here only for the sake of completeness.

INPUT: x
OUTPUT: $\mathsf{w_h}(x)$

1. $x \leftarrow x - ((x \gg 1) \wedge \texttt{0x55555555L})$;
2. $x \leftarrow (x \wedge \texttt{0x33333333L}) + ((x \gg 2) \wedge \texttt{0x33333333L})$;
3. $x \leftarrow (x + (x \gg 4)) \wedge \texttt{0x0F0F0F0FL}$;
4. $x \leftarrow x + (x \gg 8)$;
5. $x \leftarrow (x + (x \gg 16)) \wedge \texttt{0x0000003FL}$;
6. Return x;

Additional time-space trade-offs are possible in calculating the Hamming weight. If $n = cm$, then one can precompute 2^c values of $\mathsf{w_h}(i)$, $0 \leq i < 2^c$, and then find $\mathsf{w_h}(x)$ by doing $m = n/c$ table look-ups. This method is faster than the method described in the previous paragraph if $m \leq \log_2 n$, which is the case if $n = 32$ and $m \in \{8, 16\}$. However, it also requires more memory. While we do not discuss this method hereafter, our implementations use it, since it offers better performance on 32-bit processors.

$$
\begin{aligned}
x &= 00001000001100110000010101010100\ ,\\
y &= 01000000000000010110110001110100\ ,\\
\mathsf{aop}(x) &= 00001000001000100000010101010100\ ,\\
\mathsf{aop}^r(x) &= 00001000000100010000010101010100\ ,\\
C(x,y) &= 00000000000000001000001001001010\ ,\\
C^r(x,y) &= 00000000000000001000010010010100\ .
\end{aligned}
$$

Fig. 1. A pair (x, y) with corresponding values $\mathsf{aop}(x)$, $\mathsf{aop}^r(x)$, $C(x, y)$ and $C^r(x, y)$. Here, for example, $\mathsf{aop}(x)_{27} = 1$ since $1 = x_{27} \neq x_{28}$ and $28 - 27 = 1$ is odd. On the other hand, $C^r(x, y)_4 = 1$ since $x_4 = y_4 \neq x_3 = y_3 \neq x_2 = y_2 \neq x_1 = y_1 = x_0 = y_0$, and $4 - 0$ is even. Since $\ell_5 = 0$, we could have taken also $C(x, y)_5 = 1$

Interestingly, many ancient and modern power architectures have a special machine-level "cryptanalyst's" instruction for $\mathsf{w_h}$ (mostly known as the *population count* instruction): SADD on the Mark I (*sic*), CXi Xj on the CDC Cyber series, Ai PSj on the Cray X-MP, VPCNT on the NEC SX-4, CTPOP on the Alpha 21264, POPC on the Ultra SPARC, POPCNT on the Intel IA64, etc. In principle, we could incorporate in our model a unit-time population count instruction, then several later presented algorithms would run in constant time. However, since there is no population count instruction on most of the other architectures (especially on the widespread Intel IA32 platform), we have decided not include it in the set of primitive operations. Moreover, the complexity of population count does not significantly influence the (average-case) complexity of the derived algorithms.

All-one and Common Alternation Parity. The second and third functions, important for several derived algorithms (more precisely, they are used in Algorithm 4 and Algorithm 5), are the all-one and common alternation parity of n-bit strings, defined as follows. (Note that while the Hamming weight has very many useful applications in cryptography, the functions defined in this section have never been, as far as we know, used before for any cryptographic or other purpose.)

The *all-one parity* of an n-bit number x is another n-bit number $y = \mathsf{aop}(x)$ s.t. $y_i = 1$ iff the longest sequence of consecutive one-bits $x_i x_{i+1} \ldots x_{i+j} = 11 \ldots 1$ has odd length.

The *common alternation parity* of two n-bit numbers x and y is a function $C(x, y)$ with the next properties: (1) $C(x, y)_i = 1$, if ℓ_i is even and non-zero, (2) $C(x, y)_i = 0$, if ℓ_i is odd, (3) unspecified (either 0 or 1) if $\ell_i = 0$, where ℓ_i is the length of the longest common alternating bit chain $x_i = y_i \neq x_{i+1} = y_{i+1} \neq x_{i+2} = y_{i+2} \ldots \neq x_{i+\ell_i} = y_{i+\ell_i}$, where $i + \ell_i \leq n - 1$. (In both cases, counting starts with one. E.g., if $x_i = y_i$ but $x_{i+1} \neq y_{i+1}$ then $\ell_i = 1$ and $C(x, y)_i = 0$.) W.l.o.g., we will define

$$C(x, y) := \mathsf{aop}(\neg(x \oplus y) \wedge (\neg(x \oplus y) \gg 1) \wedge (x \oplus (x \gg 1)))\ .$$

For both the all-one and common alternation parity we will also need their "duals" (denoted as aop^r and C^r), obtained by bit-reversing their arguments. That is,

Algorithm 1 Log-time algorithm for $\mathsf{aop}(x)$

INPUT: $x \in \Sigma^n$, n is a power of 2
OUTPUT: $\mathsf{aop}(x)$

1. $x[1] = x \wedge (x \gg 1)$;
2. For $i \leftarrow 2$ to $\log_2 n - 1$ do $\quad x[i] \leftarrow x[i-1] \wedge (x[i-1] \gg 2^{i-1})$;
3. $y[1] \leftarrow x \wedge \neg x[1]$;
4. For $i \leftarrow 2$ to $\log_2 n$ do $\quad y[i] \leftarrow y[i-1] \vee ((y[i-1] \gg 2^{i-1}) \wedge x[i-1])$;
5. Return $y[\log_2 n]$;

$\mathsf{aop}^r(x,y) = \mathsf{aop}(x',y')$, where $x'_i := x_{n-i}$ and $y'_i := y_{n-i}$. (See Fig. 1.) Note that for every (x,y) and i, $C(x,y)_i = 1 \Rightarrow C(x,y)_{i+1} = C(x,y)_{i-1} = 0$.

Clearly, Algorithm 1 finds the all-one parity of x in time $\Theta(\log n)$. (It is sufficient to note that $x[i]_j = 1$ if and only if the number n_j of ones in the sequence $(x_j = 1, x_{j+1} = 1, \ldots, x_{j+n_j-1} = 1, x_{j+n_j-2} = 0)$ is at least 2^i and $y[i]_j = 1$ iff n_j is an odd number not bigger than 2^j.) Therefore also $C(x,y)$ can be computed in time $\Theta(\log n)$.

4 Log-time Algorithm for Differential Probability of Addition

In this section we say that differential $\delta = (\alpha, \beta \mapsto \gamma)$ is "*good*" if $\mathsf{eq}(\alpha \ll 1, \beta \ll 1, \gamma \ll 1) \wedge (\mathsf{xor}(\alpha,\beta,\gamma) \oplus (\alpha \ll 1)) = 0$. Alternatively, δ is not "good" iff for some $i \in [0, n-1]$, $\alpha_{i-1} = \beta_{i-1} = \gamma_{i-1} \neq \alpha_i \oplus \beta_i \oplus \gamma_i$. (Remember that $\alpha_{-1} = \beta_{-1} = \gamma_{-1} = 0$.) The next algorithm has a simple linear-time version, suitable for "manual cryptanalysis": (1) Check, whether δ is "good", using the second definition of "goodness"; (2) If δ is "good", count the number of positions $i \neq n-1$, s.t. the triple $(\alpha_i, \beta_i, \gamma_i)$ contains both zeros and ones.

Theorem 1. *Let $\delta = (\alpha, \beta \mapsto \gamma)$ be an arbitrary differential. Algorithm 2 returns $\mathrm{DP}^+(\delta)$ in time $\Theta(\log n)$. More precisely, it works in time $\Theta(1) + t$, where t is the time it takes to compute $\mathsf{w_h}$.*

Algorithm 2 Log-time algorithm for DP^+

INPUT: $\delta = (\alpha, \beta \mapsto \gamma)$
OUTPUT: $\mathrm{DP}^+(\delta)$

1. If $\mathsf{eq}(\alpha \ll 1, \beta \ll 1, \gamma \ll 1) \wedge (\mathsf{xor}(\alpha,\beta,\gamma) \oplus (\beta \ll 1)) \neq 0$ then return 0;
2. Return $2^{-\mathsf{w_h}(\neg \mathsf{eq}(\alpha,\beta,\gamma) \wedge \mathsf{mask}(n-1))}$;

Rest of this subsection consists of a step-by-step proof of this result, where we use the Lemma 1, i.e., that $\mathrm{DP}^+(\delta) = \mathbf{P}_{x,y}[\mathsf{carry}(x,y) \oplus \mathsf{carry}(x \oplus \alpha, y \oplus \beta) = \mathsf{xor}(\alpha,\beta,\gamma)$.

We first state and prove two auxiliary lemmas. After that we show how Theorem 1 follows from them, and present two corollaries.

Lemma 2. *Let $L(x)$ be a mapping, such that $L(0) = 0$, $L(1) = L(2) = \frac{1}{2}$ and $L(3) = 1$. Let $\alpha, \beta \in \Sigma^n$. Then $\mathbf{P}_{x,y}[\mathsf{carry}(x,y)_{i+1} \oplus \mathsf{carry}(x \oplus \alpha, y \oplus \beta)_{i+1} = 1 | \alpha_i + \beta_i + \Delta c_i = j] = L(j)$.*

Proof. We denote $c = \mathsf{carry}(x,y)$ and $c^* = \mathsf{carry}(x \oplus \alpha, y \oplus \beta)$, where x and y are understood from the context. Let also $\Delta c = c \oplus c^*$. By the definition of carry, $\Delta c_{i+1} = (x_i \wedge y_i) \oplus (x_i \wedge c_i) \oplus (y_i \wedge c_i) \oplus ((x \oplus \alpha)_i \wedge (y \oplus \beta)_i) \oplus ((x \oplus \alpha)_i \wedge c_i^*) \oplus ((y \oplus \beta)_i \wedge c_i^*)$. This formula for Δc_{i+1} is symmetric in the three pairs (x_i, α_i), (y_i, β_i) and $(c_i, \Delta c_i)$. Hence, the function $f(\alpha_i, \beta_i, \Delta c_i) := \mathbf{P}_{x_i,y_i,c_i}[\Delta c_{i+1} = 1]$ is symmetric, and therefore f is a function of $\alpha_i + \beta_i + \Delta c_i$, $f(j) = \mathbf{P}_{x_i,y_i,c_i}[\Delta c_{i+1} = 1 | \alpha_i + \beta_i + \Delta c_i = j]$. One can now prove that $\mathbf{P}_{x_i,y_i}[\Delta c_{i+1} = 1 | \alpha_i + \beta_i + \Delta c_i = j] = L(j)$ for any $0 \le j \le 3$, and for any value of $c_i \in \{0,1\}$. For example, $\mathbf{P}_{x_i,y_i}[\Delta c_{i+1} = 1 | \alpha_i + \beta_i + \Delta c_i = 1] = \mathbf{P}_{x_i,y_i}[\Delta c_{i+1} = 1 | (\alpha_i, \beta_i, \Delta c_i) = (0,0,1)] = \mathbf{P}_{x_i,y_i}[(x_i \wedge c_i) \oplus (y_i \wedge c_i) \oplus (x_i \wedge \neg c_i) \oplus (x_i \wedge \neg c_i) = 1] = \mathbf{P}_{x_i,y_i}[x_i = y_i] = \frac{1}{2}$. The claim follows since $\mathbf{P}_{x,y}[\Delta c_{i+1}] = \mathbf{P}_{x_i,y_i}[\Delta c_{i+1}]$. □

Lemma 3. *1) Every possible differential is "good".*
2) Let $\delta = (\alpha, \beta \mapsto \gamma)$ be "good". If $i \in [0, n-1]$, then $\mathbf{P}_{x,y}[\mathsf{carry}(x,y)_i \oplus \mathsf{carry}(x \oplus \alpha, y \oplus \beta)_i = 1 | \alpha_{i-1} + \beta_{i-1} + \gamma_{i-1} = j] = L(j)$. In particular, $\mathbf{P}_{x,y}[\mathsf{carry}(x,y)_0 \oplus \mathsf{carry}(x \oplus \alpha, y \oplus \beta)_0 = 0] = 1$.

Proof. 1) Let δ be possible but not "good". By Lemma 1, there exists an i and a pair (x,y), s.t. $\mathsf{carry}(x,y)_{i+1} \oplus \mathsf{carry}(x \oplus \alpha, y \oplus \beta)_{i+1} = \mathsf{xor}(\alpha, \beta, \gamma)_{i+1} \neq \alpha_i = \beta_i = \gamma_i$. Note that then $\mathsf{xor}(\alpha, \beta, \gamma)_i = \gamma_i$. But by Lemma 2, $\mathbf{P}_{x_i,y_i}[\mathsf{carry}(x,y)_{i+1} \oplus \mathsf{carry}(x \oplus \alpha, y \oplus \beta)_{i+1} \neq \gamma_i | \alpha_i = \beta_i = \gamma_i] = 0$, which is a contradiction.

2) Let δ be "good". We prove the theorem by induction on i, by simultaneously proving the induction invariant $\mathbf{P}_{x,y}[\mathsf{carry}(x,y) \oplus \mathsf{carry}(x \oplus \alpha, y \oplus \beta) = \mathsf{xor}(\alpha, \beta, \gamma) \pmod{2^i}] > 0$. BASE ($i = 0$). Straightforward from Property 1 and the definition of a "good" differential. STEP ($i + 1 > 0$). We assume that the invariant is true for i. In particular, there exists a pair (x,y), s.t. $\Delta c_{i-1} = \mathsf{xor}(\alpha, \beta, \gamma)_{i-1}$, where $\Delta c = \mathsf{carry}(x,y) \oplus \mathsf{carry}(x \oplus \alpha, y \oplus \beta)$. Then, by Lemma 2, $L(j) = \mathbf{P}_{x,y}[\Delta c_i = 1 | \alpha_{i-1} + \beta_{i-1} + \Delta c_{i-1} = j] = \mathbf{P}_{x,y}[\Delta c_i = 1 | \alpha_{i-1} + \beta_{i-1} + \mathsf{xor}(\alpha, \beta, \gamma)_{i-1} = j] = \mathbf{P}_{x,y}[\Delta c_i = 1 | \alpha_{i-1} + \beta_{i-1} + \gamma_{i-1} = j]$, where the last equation follows from the easily verifiable equality $L(a_1 + a_2 + a_3) = L(a_1 + a_2 + \mathsf{xor}(a_1, a_2, a_3))$, for every $a_1, a_2, a_3 \in \Sigma$. This proves the theorem claim for i. The invariant for i, $\Delta c_i = \mathsf{xor}(\alpha, \beta, \gamma)_i$, follows from that and the "goodness" of δ. □

Proof (Theorem 1). First, δ is "good" iff it is possible. (The "if" part follows from the first claim of Lemma 3. The "only if" part follows from the second claim of Lemma 3 and the definition of a "good" differential.) Let δ be possible. Then, by Lemma 1, $\mathrm{DP}^+(\delta) = \prod_{i=0}^{n-2} \mathbf{P}_{x,y}[\mathsf{carry}(x,y)_i \oplus \mathsf{carry}(x \oplus \alpha, y \oplus \beta)_i = \mathsf{xor}(\alpha, \beta, \gamma)_i]$. By Lemma 2, $\mathbf{P}_{x,y}[\mathsf{carry}(x,y)_i \oplus \mathsf{carry}(x \oplus \alpha, y \oplus \beta)_i = \mathsf{xor}(\alpha, \beta, \gamma)_i]$ is either 1 or $\frac{1}{2}$, depending on whether $\alpha_{i-1} = \beta_{i-1} = \gamma_{i-1}$ or not. (This probability cannot be 0, since δ is possible and hence "good".) Therefore, $\mathrm{DP}^+(\delta) = 2^{-\sum_{i=0}^{n-2} \neg \mathsf{eq}(\alpha, \beta, \gamma)_i} =$

$2^{-\mathsf{w_h}(\neg\mathsf{eq}(\alpha,\beta,\gamma)\wedge\mathsf{mask}(n-1))}$, as required. Finally, the only non-constant time computation is that of Hamming weight. □

Note that *technically*, for Algorithm 2 to be log-time it would have to return (say) -1 if the differential is impossible, or $\log_2 \mathrm{DP}^+(\delta)$, if it is not. (The other valid possibility would be to include data-dependent shifts in the set of unit-cost operations.)

The next two corollaries follow straightforwardly from Algorithm 2

Corollary 1. DP^+ *is symmetric in its arguments. That is, for an arbitrary triple* (α,β,γ), $\mathrm{DP}^+(\alpha,\beta\mapsto\gamma) = \mathrm{DP}^+(\beta,\alpha\mapsto\gamma) = \mathrm{DP}^+(\alpha,\gamma\mapsto\beta)$. *Therefore, in particular,* $\max_\alpha \mathrm{DP}^+(\alpha,\beta\mapsto\gamma) = \max_\beta \mathrm{DP}^+(\alpha,\beta\mapsto\gamma) = \max_\gamma \mathrm{DP}^+(\alpha,\beta\mapsto\gamma)$.

Corollary 2. *1) [Conjecture 1, [AKM98].] Let* $\alpha+\beta = \alpha'+\beta'$ *and* $\alpha\oplus\beta = \alpha'\oplus\beta'$. *Then for every* γ, $\mathrm{DP}^+(\alpha,\beta\mapsto\gamma) = \mathrm{DP}^+(\alpha',\beta'\mapsto\gamma)$. *2) For every* α, β, γ, $\mathrm{DP}^+(\alpha,\beta\mapsto\gamma) = \mathrm{DP}^+(\alpha\wedge\beta,\alpha\vee\beta\mapsto\gamma)$.

Proof. We say that (α,β) and (α',β') are *equivalent*, if $\{\alpha_i,\beta_i\} = \{\alpha'_i,\beta'_i\}$ for $i < n-1$, and $\alpha_{n-1}\oplus\beta_{n-1} = \alpha'_{n-1}\oplus\beta_{n-1}$. If (α,β) and (α',β') are equivalent then $\mathrm{DP}^+(\alpha,\beta\mapsto\gamma) = \mathrm{DP}^+(\alpha',\beta'\mapsto\gamma)$ by the structure of Algorithm 2.

1) The corresponding carries $c = \mathsf{carry}(\alpha,\beta)$ and $c' = \mathsf{carry}(\alpha',\beta')$ are equal, since $c = (\alpha\oplus\beta)\oplus(\alpha+\beta) = (\alpha'\oplus\beta')\oplus(\alpha'+\beta') = c'$. Therefore, $\alpha+\beta = \alpha'+\beta'$ and $\alpha\oplus\beta = \alpha'\oplus\beta'$ iff (α,β) and (α',β') are equivalnet.

2) The second claim is straightforward, since (α,β) and $(\alpha\wedge\beta,\alpha\vee\beta)$ are equivalent for any α and β. □

Note that a pair (x,y) is equivalent to $2^{1+\mathsf{w_h}((x\oplus y)\wedge\mathsf{mask}(n-1))}$ different pairs (x^*,y^*). In [DGV93, Sect. 2.3] it was briefly mentioned that the number of such pairs is not more than $2^{\mathsf{w_h}(x\oplus y)}$; this result was used to cryptanalyse IDEA. The second claim carries unexpected connotations with the well known fact that $\alpha+\alpha' = (\alpha\wedge\alpha')+(\alpha\vee\alpha')$.

5 Statistical Properties of Differential Probability

Note that Algorithm 2 has two principal steps. The first step is a constant-time check of whether the differential $\delta = (\alpha,\beta\mapsto\gamma)$ is impossible (i.e., whether $\mathrm{DP}^+(\delta) = 0$). The second step, executed only if δ is possible, computes in log-time the Hamming weight of an n-bit string. The structure of this algorithm raises an immediate question of what is the density $\mathbf{P}_\delta[\mathrm{DP}^+(\delta) \neq 0]$ of the possible differentials, since its average-case complexity (where the average is taken over uniformly and random chosen differentials δ) is $\Theta(\mathbf{P}_\delta[\mathrm{DP}^+(\delta)=0] + \mathbf{P}_\delta[\mathrm{DP}^+(\delta)\neq 0]\cdot\log n)$. This is one (but certainly not the only or the most important) motivation for the current section.

Let $X : \delta \mapsto \mathrm{DP}^+(\delta)$ be a uniformly random variable. We next calculate the exact probabilities $\mathbf{P}[X = i]$ for any i. From the results we can directly derive the distribution of X. Knowing the distribution, one can, by using standard probabilistic tools, calculate the values of many other interesting probabilistic properties like the probabilities $\mathbf{P}[X > i]$ for any i.

Theorem 2. *1) [Conjecture 2, [AKM98].]* $\mathbf{P}[X \neq 0] = \frac{1}{2} \cdot \left(\frac{7}{8}\right)^{n-1}$.
2) Let $0 \leq k < n$. *Then* $\mathbf{P}[X = 2^{-k}] = 2^{2+k-3n} \cdot 3^k \cdot \binom{n-1}{k} = \frac{1}{2} \cdot \left(\frac{7}{8}\right)^{n-1} \cdot b\left(k; n-1, \frac{6}{7}\right)$.

Proof. Let $\delta = (\alpha, \beta \mapsto \gamma)$ be an arbitrary differential and let $e = \mathsf{eq}(\alpha, \beta, \gamma)$, $e' = \mathsf{eq}(\alpha \ll 1, \beta \ll 1, \gamma \ll 1)$ and $x = \mathsf{xor}(\alpha, \beta, \gamma) \oplus (\alpha \ll 1))$ be convenient shorthands. Since α, β and γ are mutually independent, e and x (and also e' and x) are pairwise independent.

1) From Theorem 1, $\mathbf{P}[X \neq 0] = \mathbf{P}_\delta[e' \wedge x = 0] = \prod_{i=0}^{n-1}(1 - \mathbf{P}_\delta[e'_i = 1, x_i = 1]) = \prod_{i=0}^{n-1}(1 - \mathbf{P}_\delta[e'_i = 1] \cdot \mathbf{P}_\delta[x_i = 1]) = \left(1 - 1 \cdot \frac{1}{2}\right) \cdot \prod_{i=1}^{n-1}\left(1 - \frac{1}{4} \cdot \frac{1}{2}\right) = \frac{1}{2} \cdot \left(\frac{7}{8}\right)^{n-1}$.

2) Let $m = \mathsf{mask}(n-1)$. First, clearly, for any $0 \leq k < n$, $\mathbf{P}_\delta[\mathsf{w_h}(e) = k] = \mathbf{P}_{\alpha,\beta,\gamma \in \Sigma^n}[\mathsf{w_h}(e) = k] = \left(\frac{1}{4}\right)^k \cdot \left(\frac{3}{4}\right)^{n-k} \cdot \binom{n}{k} = b\left(k; n, \frac{1}{4}\right)$ and therefore $\mathbf{P}_\delta[\mathsf{w_h}(\neg e \wedge m) = k] = b\left(n-1-k; n-1, \frac{1}{4}\right) = \left(\frac{1}{4}\right)^{n-1-k} \cdot \left(\frac{3}{4}\right)^k \cdot \binom{n-1}{k} = 2^{2-2n} \cdot 3^k \cdot \binom{n-1}{k}$.

Let A denote the event $\mathsf{w_h}(e \wedge m) = n-1-k$ and let B denote the event $e' \wedge x = 0$. Let B_i be the event $e'_i \wedge x_i = 0$. According to Algorithm 2, $\mathbf{P}[X = 2^{-k}] = \mathbf{P}_\delta[A, B] = \mathbf{P}_\delta[A] \cdot \mathbf{P}_\delta[B|A] = \mathbf{P}_\delta[A] \cdot \prod_{i=0}^{n-1} \mathbf{P}_\delta[B_i|A] = \frac{1}{2} \cdot \mathbf{P}_\delta[A] \cdot \prod_{i=1}^{n-1} \mathbf{P}_\delta[B_i|A]$, where we used the fact that $e'_0 = 1$.

Now, if $i > 0$ then $\mathbf{P}_\delta[B_i|e'_i = 1] = \mathbf{P}_\delta[x_i = 0] = \frac{1}{2}$, while $\mathbf{P}_\delta[B_i = 0|e'_i = 0] = 1$. Moreover, $e'_i = e_{i-1}$. Therefore, $\prod_{i=0}^{n-1} \mathbf{P}_\delta[B_i|A] = \left(\frac{1}{2}\right)^{n-1-k}$, and hence $\mathbf{P}[X = 2^{-k}] = \frac{1}{2} \cdot \mathbf{P}_\delta[A] \cdot \prod_{i=0}^{n-1} \mathbf{P}_\delta[B_i|A] = \frac{1}{2} \cdot b\left(n-1-k; n-1, \frac{1}{4}\right) \cdot \left(\frac{1}{2}\right)^{n-1-k} = \frac{1}{2} \cdot 2^{2-2n} \cdot 3^k \cdot \binom{n-1}{k} \cdot 2^{1+k-n} = 2^{2+k-3n} \cdot 3^k \cdot \binom{n-1}{k} = \frac{1}{2} \cdot \left(\frac{7}{8}\right)^{n-1} \cdot b\left(k; n-1, \frac{6}{7}\right)$. □

Corollary 3. *Algorithm 2 has average-case complexity* $\Theta(1)$.

As another corollary, $X = X_0 + X_1$, where $X_1, X_2 : \delta \mapsto \mathrm{DP}^+(\delta)$ are two random variables. X_0 has domain $\mathcal{D}(X_0) = \{\delta \in \Sigma^{3n} : \mathrm{DP}^+(\delta) = 0\}$, while X_1 has the complementary domain $\mathcal{D}(X_1) = \{\delta \in \Sigma^{3n} : \mathrm{DP}^+(\delta) \neq 0\}$. Moreover, X_0 has constant distribution (since $\mathbf{P}[X_0 = 0] = 1$), while the random variable $-\log_2 X_1$ has binomial distribution with $p = \frac{6}{7}$. Knowledge of the distribution helps to find further properties of DP^+ (e.g., the probabilities that $\mathrm{DP}^+(\delta) > 2^{-k}$) by using standard methods from probability theory.

One can double-check the correctness of Theorem 2 by verifying that $\frac{1}{2} \cdot (\frac{7}{8})^{n-1} \cdot \sum_{k=0}^{n-1} b\left(k; n-1, \frac{6}{7}\right) = \sum_{k=0}^{n-1} \mathbf{P}[X = 2^{-k}] = \mathbf{P}[X \neq 0] = \frac{1}{2} \cdot \left(\frac{7}{8}\right)^{n-1}$. Moreover, clearly $\mathbf{P}[X = 2^{-k}] = |\mathcal{D}(X_0)| \cdot \mathbf{P}[X_0 = 2^{-k}] + |\mathcal{D}(X_1)| \cdot \mathbf{P}[X_1 = 2^{-k}] = \frac{1}{2} \cdot \left(\frac{7}{8}\right)^{n-1} \cdot b\left(k; n-1, \frac{6}{7}\right)$, which agrees with Theorem 2.

We next compute the variance of X. Clearly, $\mathbf{E}[X] = \sum_{k=0}^{n-1} 2^{-k} \mathbf{P}[X = 2^{-k}] = 2^{-n}$, and therefore $\mathbf{E}[X]^2 = 2^{-2n}$. Next, by using Theorem 2 and the basic properties of the binomial distribution, $\mathbf{E}[X^2] = 0 \cdot \mathbf{P}[X^2 = 0] + \sum_{k=0}^{n-1} 2^{-2k} \cdot \mathbf{P}[X^2 = 2^{-2k}] = \frac{1}{2} \cdot \left(\frac{7}{8}\right)^{n-1} \cdot \sum_{k=0}^{n-1} 2^{-2k} \cdot b\left(k; n-1, \frac{6}{7}\right) = \frac{1}{2} \cdot \left(\frac{5}{16}\right)^{n-1} \cdot \sum_{i=0}^{n-1} b\left(k; n-1, \frac{3}{5}\right) = \frac{1}{2} \cdot \left(\frac{5}{16}\right)^{n-1}$. Therefore, $\mathbf{Var}[X] = \frac{1}{2} \cdot \left(\frac{5}{16}\right)^{n-1} - 2^{-2n} = \frac{1}{2} \cdot \left(\left(\frac{5}{16}\right)^{n-1} - \left(\frac{4}{16}\right)^{n-1}\right)$.

Note that the density of possible differentials $\mathbf{P}[X \neq 0]$ is exponentially small in n. This can be contrasted with a result of O'Connor [O'C95] that a randomly selected n-bit

Algorithm 3 Algorithm that finds all γ-s, s.t. $\mathrm{DP}^{+}(\alpha, \beta \mapsto \gamma) = \mathrm{DP}^{+}_{\mathsf{max}}(\alpha, \beta)$

INPUT: (α, β)
OUTPUT: All (α, β)-optimal output differences γ

1. $\gamma_0 \leftarrow \alpha_0 \oplus \beta_0$;
2. $p \leftarrow C(\alpha, \beta)$;
3. For $i \leftarrow 1$ to $n-1$ do
   If $\alpha_{i-1} = \beta_{i-1} = \gamma_{i-1}$ then $\gamma_i \leftarrow \alpha_i \oplus \beta_i \oplus \alpha_{i-1}$
   else if $i = n-1$ or $\alpha_i \neq \beta_i$ or $p_i = 1$ then $\gamma_i \leftarrow \{0, 1\}$
   else $\gamma_i \leftarrow \alpha_i$;
4. Return γ.

permutation has a fraction of $1 - e^{-1/2} \approx 0.4$ impossible differentials, independently of the choice of n. Moreover, a randomly selected n-bit composite permutation [O'C93], controlled by an n-bit string, has a negligible fraction $\approx 2^{3n}/e^{2^n - 1}$ of impossible differentials.

6 Algorithms for Finding Good Differentials of Addition

The last section described methods for computing the probability that a randomly picked differential δ has high differential probability. While this alone might give rise to successful differential attacks, it would be nice to have an efficient deterministic algorithm for finding differentials with high differential probability. This section gives some relevant algorithms for this.

6.1 Linear-time Algorithm for $\mathrm{DP}^{+}_{\mathsf{max}}$

In this subsection, we will describe an algorithm that, given an input difference (α, β), finds all output differences γ, for which $\mathrm{DP}^{+}(\alpha, \beta \mapsto \gamma)$ is equal to the *maximum differential probability of addition*, $\mathrm{DP}^{+}_{\mathsf{max}}(\alpha, \beta) := \max_{\gamma} \mathrm{DP}^{+}(\alpha, \beta \mapsto \gamma)$. (By Corollary 1, we would get exactly the same result when maximizing the differential probability under α or β.) We say that such γ is *(α, β)-optimal*. Note that when an (α, β)-optimal γ is known, the maximum differential probability can be found by applying Algorithm 2 to $\delta = (\alpha, \beta \mapsto \gamma)$. Moreover, similar algorithms can be used to find "near-optimal" γ-s, where $\log_2 \mathrm{DP}^{+}(\alpha, \beta \mapsto \gamma)$ is only slightly smaller than $\log_2 \mathrm{DP}^{+}_{\mathsf{max}}(\alpha, \beta)$.

Theorem 3. *Algorithm 3 returns all (α, β)-optimal output differences γ.*

Proof (Sketch). First, we say that position i is *bad* if $\mathsf{eq}(\alpha, \beta, \gamma)_i = 0$. According to Theorem 1, γ is (α, β)-optimal if it is chosen so that (1) for every $i \geq 0$, if $\mathsf{eq}(\alpha, \beta, \gamma)_{i-1} = 1$ then $\mathsf{xor}(\alpha_i, \beta_i, \gamma_i) = \alpha_{i-1}$, and (2) the number of bad positions i is the least among all such output differences γ', for which $(\alpha, \beta \mapsto \gamma')$ is possible. For

Algorithm 4 A log-time algorithm that finds an (α, β)-optimal γ

INPUT: (α, β)
OUTPUT: An (α, β)-optimal γ

1. $r \leftarrow \alpha \wedge 1$;
2. $e \leftarrow \neg(\alpha \oplus \beta) \wedge \neg r$;
3. $a \leftarrow e \wedge (e \ll 1) \wedge (\alpha \oplus (\alpha \ll 1))$;
4. $p \leftarrow \mathtt{aop}^r(a)$;
5. $a \leftarrow (a \vee (a \gg 1)) \wedge \neg r$;
6. $b \leftarrow (a \vee e) \ll 1$;
7. $\gamma \leftarrow ((\alpha \oplus p) \wedge a) \vee ((\alpha \oplus \beta \oplus (\alpha \ll 1)) \wedge \neg a \wedge b) \vee (\alpha \wedge \neg a \wedge \neg b)$;
8. $\gamma \leftarrow (\gamma \wedge \neg 1) \vee ((\alpha \oplus \beta) \wedge 1)$;
9. Return γ.

achieving (1) we must first fix $\gamma_0 \leftarrow \alpha_0 \oplus \beta_0$, and after that recursively guarantee that γ_i obtains the predicted value whenever $\alpha_{i-1} = \beta_{i-1} = \gamma_{i-1}$.

This, and minimizing the number of bad i-s can be done recursively for every $0 \leq i \leq n-1$, starting from $i = 0$. If $\alpha_i \neq \beta_i$ then i is bad independently of the value of $\gamma_i \in \{0, 1\}$. Moreover, either choice of γ places no restriction on choosing γ_{i+1}. This means that we can assign either $\gamma_i \leftarrow 0$ or $\gamma_i \leftarrow 1$.

The situation is more complicated if $\alpha_i = \beta_i$. Intuitively, if $\ell_i = 2k > 0$ is even, then the choice $\gamma_i \leftarrow \alpha_i$ (as compared to the choice $\gamma_i \leftarrow \neg\alpha_i$) will result in k bad positions $(i, i+2, \ldots, i+2k-2)$ instead of k bad positions $(i+1, i+3, \ldots, i+2k-1)$. Thus these two choices are equal. On the other hand, if $\ell_i = 2k+1 > 0$, then the choice $\gamma_i \leftarrow \alpha_i$ would result in k bad positions compared to $k+1$ when $\gamma_i \leftarrow \alpha_i$, and hence is to be preferred over the second one. We leave the full details of the proof to the reader. □

A linear-time algorithm that finds one (α, β)-optimal γ can be derived from Algorithm 3 straightforwardly, by assigning $\gamma_i \leftarrow \alpha_i$ whenever $\mathsf{eq}(\alpha, \beta, \gamma)_{i-1} = 0$.

As an example, let us look at the case $n = 16$, $\alpha = \mathtt{0x5254}$ and $\beta = \mathtt{0x1A45}$. Then $C(\alpha, \beta) = \mathtt{0x1244}$, and by Algorithm 3 the set of (α, β)-optimal values is equal to $2^{15}\Sigma + 2^{12}P + 2^{11}\Sigma + 2^4\Sigma + 1$, where $P = \{0, 3, 7\}$, and $\Sigma = \{0, 1\}$, as always. Therefore, for example, $\mathrm{DP}^+_{\mathsf{max}}(\mathtt{0x5254}, \mathtt{0x1A45}) = \mathrm{DP}^+(\mathtt{0x5254}, \mathtt{0x1A45} \mapsto \mathtt{0x7011}) = 2^{-8}$.

6.2 Log-time Algorithm for $\mathrm{DP}^+_{\mathsf{max}}$

For a log-time algorithm we need a somewhat different approach, since in Algorithm 3 the value of γ_i depends on that of γ_{i-1}. However, luckily for us, γ_i only depends on γ_{i-1} if $\mathsf{eq}(\alpha, \beta, \gamma)_{i-1} = 1$, and as seen from the proof of Theorem 3, in many cases we can choose the output difference γ_{i-1} so that $\mathsf{eq}(\alpha, \beta, \gamma)_{i-1} = 0$!

Moreover, the positions where $\mathsf{eq}(\alpha, \beta, \gamma)_{i-1} = 1$ must hold are easily detected. Namely (see Algorithm 3), if (1) $i = 0$ and $\alpha_i = \beta_i = 0$, or (2) $i > 0$ and $\alpha_i = \beta_i$ but $p_i = 0$. Accordingly, we can replace the condition $\mathsf{eq}(\alpha, \beta, \gamma)_{i-1} = 1$ with the

Algorithm 5 A log-time algorithm for $\mathrm{DP}^{+}_{\mathsf{2max}}(\alpha)$

INPUT: α
OUTPUT: $\mathrm{DP}^{+}_{\mathsf{2max}}(\alpha)$

1. Return $2^{-\mathsf{w_h}(C^r(\alpha,\alpha)\wedge\mathsf{mask}(n-1))}$.

condition $\neg(\alpha_i \oplus \beta_i) \wedge p_i$, and take additional care if i is small. By noting how the values p_i are computed, one can prove that

Theorem 4. *Algorithm 4 finds an (α, β)-optimal γ.*

Proof (Sketch, many details omitted). First, the value of p computed in the step 4 is "approximately" equal to $C^r(\alpha, \beta)$, with some additional care taken about the lowest bits. Let ℓ^r be the bit-reverse of ℓ (i.e., ℓ^r_i is equal to the length of longest common alternating chain $\alpha_i = \beta_i \neq \alpha_{i-1} = \beta_{i-1} \neq \ldots$). Step 7 computes γ_i (again, "approximately") as (1) $\gamma_i \leftarrow \alpha_i \oplus p_i$, if $\alpha_i = \beta_i$, (2) $\gamma_i \leftarrow \alpha_i \oplus \beta_i \oplus \alpha_{i-1}$ if $\alpha_i \neq \beta_i$ but $\mathsf{eq}(\alpha, \beta, \gamma)_{i-1} = 1$ and (3) $\gamma_i \leftarrow \alpha_i$ if $\alpha_i \neq \beta_i$ and $\mathsf{eq}(\alpha, \beta, \gamma)_{i-1} = 0$. Since the two last cases are sound, according to Algorithm 3, we are now left to prove that the first choice makes γ optimal.

But this choice means that in every maximum-length common alternating bit chain $\alpha_i = \beta_i \neq \alpha_{i-1} = \beta_{i-1} \neq \ldots \neq \alpha_{i-\ell^r_i+1} = \beta_{i-\ell^r_i+1}$, Algorithm 4 chooses all bits γ_j, $j \in [i - \ell^r_i + 1, i]$, to be equal to $\alpha_{i-\ell^r_i+1} = \beta_{i-\ell^r_i+1}$. By approximately the same arguments as in the proof of Theorem 3, this choice gives rise to $\lfloor \ell_i/2 \rfloor$ bad bit positions in the fragment $[i - \ell^r_i + 1, i]$; every other choice of bits γ_j would result in at least as many bad positions. Moreover, since $\neg(\alpha_{i+1} = \beta_{i+1} \neq \alpha_i = \beta_i)$, it has to be the case that either $\alpha_{i+1} \neq \beta_{i+1}$ or $\alpha_{i+1} = \beta_{i+1} = \alpha_i = \beta_i$. In the first case, both choices of γ_i make $i + 1$ bad. In the second case we must later take $\gamma_{i+1} \leftarrow \alpha_{i+1}$, which makes $i + 1$ good, and enables us to start the next fragment from the position $i + 1$. (Intuitively, this last fact is also the reason why aop^r is here preferred over aop.) □

Note that Biham and Shamir used the fact $\mathrm{DP}^{+}(\alpha, \beta \mapsto \alpha \oplus \beta) = 2^{-\mathsf{w_h}((\alpha\vee\beta)\wedge\mathsf{mask}(n-1))}$ in their differential cryptanalysis of FEAL in [BS91b]. Often this value is significantly smaller than the maximum differential probability $\mathrm{DP}^{+}_{\mathsf{max}}(\alpha, \beta)$. For example, if $\alpha = \beta = 2^n - 1$, then $\mathrm{DP}^{+}_{\mathsf{max}}(\alpha, \beta) = \frac{1}{2}$, while $\mathrm{DP}^{+}(\alpha, \beta \mapsto \alpha \oplus \beta) = \mathrm{DP}^{+}(\alpha, \beta \mapsto 0) = 2^{n-1}$. However, since FEAL only uses 8-bit addition, it is possible to find (α, β)-optimal output differences γ by exhaustive search. This has been done, for example, in [AKM98].

6.3 Log-time Algorithm for Double-Maximum Differential Probability

We next show that the *double-maximum differential probability*

$$\mathrm{DP}^{+}_{\mathsf{2max}}(\alpha) := \max_{\beta,\gamma} \mathrm{DP}^{+}(\alpha, \beta \mapsto \gamma) = \max_{\beta} \mathrm{DP}^{+}_{\mathsf{max}}(\alpha, \beta)$$

of addition can be computed in time $\Theta(\log n)$. (As seen from Algorithm 3, $\mathrm{DP}^{+}_{\mathsf{max}}(\alpha, \beta)$ is a symmetric function and hence $\mathrm{DP}^{+}_{\mathsf{2max}}(\alpha)$ is equal to $\mathrm{DP}^{+}(\alpha, \beta \mapsto \gamma)$ maximized under any two of its three arguments.) In particular, the next theorem shows

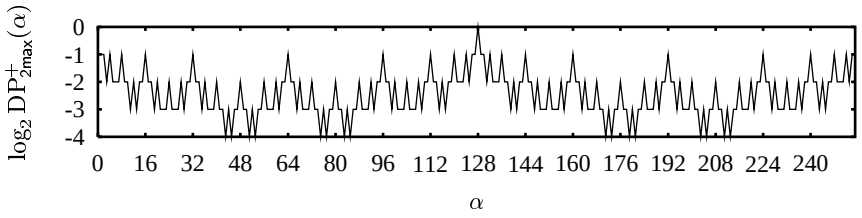

Fig. 2. Tabulation of values $\log_2 \mathrm{DP}^+_{\mathsf{2max}}(\alpha)$, $0 \le \alpha \le 255$, for $n = 8$. For example, $\mathrm{DP}^+_{\mathsf{2max}}(64) = \frac{1}{2}$ and $\mathrm{DP}^+_{\mathsf{2max}}(53) = 2^{-4}$

that $\mathrm{DP}^+_{\mathsf{2max}}(\alpha)$ is equal to the (more relevant for the DC) value $\max_{\beta \neq 0} \mathrm{DP}^+_{\mathsf{max}}(\alpha, \beta)$ whenever $\alpha \neq 0$. Note that the naive algorithm for the same problem works in time $\Omega(2^{4n})$, which makes it practically infeasible even for $n = 16$.

Theorem 5. *For every $\alpha \in \Sigma^n$, Algorithm 5 computes $\mathrm{DP}^+_{\mathsf{2max}}(\alpha)$ in time $\Theta(\log n)$.*

Proof (Sketch). By the same arguments as in the proof of previous theorem, given inputs (α, α), the value $\gamma{:=}\alpha \oplus (C^r(\alpha, \alpha) \wedge \mathsf{mask}(n-1))$ is (α, α)-optimal. We now prove by contradiction that $\mathrm{DP}^+_{\mathsf{2max}}(\alpha) = \mathrm{DP}^+_{\mathsf{max}}(\alpha, \alpha)$. Let $\beta \neq \alpha$ and γ' be such that $\mathrm{DP}^+(\alpha, \beta \mapsto \gamma') > \mathrm{DP}^+_{\mathsf{max}}(\alpha, \alpha)$. By Algorithms 2 and 4, there is an $i < n-1$ such that $\mathsf{eq}(\alpha, \beta, \gamma')_i = 1$ and $C^r(\alpha, \alpha)_i = 1$. But then, on the other hand, since the differential $(\alpha, \beta \mapsto \gamma')$ is possible and $C^r(\alpha, \alpha)_i = 1$, it is also the case that $\mathsf{eq}(\alpha, \beta, \gamma')_{i-1} = 0$. Since $C^r(\alpha, \alpha)_i = 1 \Rightarrow C^r(\alpha, \alpha)_{i-1} = 0$, we have also that $C^r(\alpha, \alpha)_{i-1} = 0$. Therefore $\mathrm{DP}^+(\alpha, \beta \mapsto \gamma') \le \mathrm{DP}^+_{\mathsf{max}}(\alpha, \alpha)$. □

Straightforwardly, Theorem 5 helps to find many interesting properties of $\mathrm{DP}^+_{\mathsf{2max}}$. For example, $\mathrm{DP}^+_{\mathsf{2max}}(\alpha) = 1$ iff $\alpha \wedge (2^{n-1} - 1) = 0$, and $\min_\alpha \mathrm{DP}^+_{\mathsf{2max}}(\alpha) = 2^{-n/2}$. Another consequence is that $\mathrm{DP}^+_{\mathsf{2max}}(\alpha) = \frac{1}{2}$ iff (1) $\alpha \wedge (2^{n-1} - 1) = -2^s + 1$ for some $0 \le s < n$, or (2) $\alpha \wedge (2^{n-1} - 1) = 2^s$ for some $0 \le s < n-1$. For better understanding, all values of $\mathrm{DP}^+_{\mathsf{2max}}(\alpha)$, $n = 8$, are depicted in Fig. 2.

Once again, our results may be compared with the results in [O'C93,O'C95] that show that for an n-bit permutation (resp. composite permutation, controlled by an n-bit string) the expected probability of the maximum nonzero differential is $\le n/2^{n-1}$ (resp. $\approx 2^{-n}$).

Further Work and Acknowledgments

While we leave practical applications of our results as an open question, we note that our results have already been used in [MY00] for *truncated* differential cryptanalysis of Twofish.

The current work bases somewhat on [Mor00], that had a (correct) linear-time algorithm for DP^+, but with an incorrect proof. We would like to thank Eli Biham for notifying us about the results presented in the full version of [BS91b].

References

[AKM98] Kazumaro Aoki, Kunio Kobayashi, and Shiho Moriai. The Best Differential Characteristic Search of FEAL. *IEICE Trans. Fundamentals*, E81-A(1):98–104, January 1998.

[Ber92] Thomas A. Berson. Differential Cryptanalysis Mod 2^{32} with Applications to MD5. In Ernest F. Brickell, editor, *Advances in Cryptology—CRYPTO '92*, volume 740 of *Lecture Notes in Computer Science*, pages 71–80. Springer-Verlag, 1993, 16–20 August 1992.

[BS91a] Eli Biham and Adi Shamir. Differential Cryptanalysis of DES-like Cryptosystems. *Journal of Cryptology*, 4(1):3–72, 1991.

[BS91b] Eli Biham and Adi Shamir. Differential Cryptanalysis of Feal and N-Hash. In Donald W. Davies, editor, *Advances on Cryptology — EUROCRYPT '91*, volume 547 of *Lecture Notes in Computer Science*, pages 1–16, Brighton, UK, April 1991. Springer-Verlag. Full version available from `http://www.cs.technion.ac.il/~biham/`, as of April 2001.

[Dae95] Joan Daemen. *Cipher and Hash Function Design. Strategies based on linear and differential cryptanalysis*. PhD thesis, Katholieke Universiteit Leuven, 1995.

[DGV93] Joan Daemen, René Govaerts, and Joos Vandewalle. Cryptanalysis of 2.5 Rounds of IDEA. Technical Report 1, ESAT-COSIC, 1993.

[Knu99] Lars Knudsen. Some Thoughts on the AES Process. Public Comment to the AES First Round, 15 April 1999. Available from `http://www.ii.uib.no/~larsr/serpent/`, as of April 2001.

[LMM91] Xuejia Lai, James L. Massey, and Sean Murphy. Markov Ciphers and Differential Cryptanalysis. In Donald W. Davies, editor, *Advances on Cryptology — EUROCRYPT '91*, volume 547 of *Lecture Notes in Computer Science*, pages 17–38, Brighton, UK, April 1991. Springer-Verlag.

[Miy98] Hiroshi Miyano. Addend Dependency of Differential/Linear Probability of Addition. *IEICE Trans. Fundamentals*, E81-A(1):106–109, January 1998.

[Mor00] Shiho Moriai. Cryptanalysis of Twofish (I). In *The Symposium on Cryptography and Information Security*, Okinawa, Japan, 26–28 January 2000. In Japanese.

[MY00] Shiho Moriai and Yiqun Lisa Yin. Cryptanalysis of Twofish (II). Technical report, IEICE, ISEC2000-38, July 2000.

[NK95] Kaisa Nyberg and Lars Knudsen. Provable Security Against a Differential Attack. *Journal of Cryptology*, 8(1):27–37, 1995.

[O'C93] Luke J. O'Connor. On the Distribution of Characteristics in Composite Permutations. In Douglas R. Stinson, editor, *Advances on Cryptology — CRYPTO '93*, volume 773 of *Lecture Notes in Computer Science*, pages 403–412, Santa Barbara, USA, August 1993. Springer-Verlag.

[O'C95] Luke O'Connor. On the Distribution of Characteristics in Bijective Mappings. *Journal of Cryptology*, 8(2):67–86, 1995.