A shortened version of this paper appears under a slightly different title
in the proceedings of *ACISP 2005* (Colin Boyd and Juan Gonzalez, eds.), LNCS, Springer-Verlag.

Solving Systems of Differential Equations of Addition and Cryptanalysis of the Helix Cipher*

Souradyuti Paul Bart Preneel

Katholieke Universiteit Leuven, Dept. ESAT/COSIC,
Kasteelpark Arenberg 10,
B–3001, Leuven-Heverlee, Belgium
{Souradyuti.Paul, Bart.Preneel}@esat.kuleuven.ac.be

Abstract

Mixing addition modulo 2^n $(+)$ and exclusive-or $(\oplus)$ has a host of applications in symmetric cryptography as the operations are fast and nonlinear over GF(2). We deal with a frequently encountered equation $(x + y) \oplus ((x \oplus \alpha) + (y \oplus \beta)) = \gamma$. The difficulty of solving an arbitrary system of such equations – named *differential equations of addition* (DEA) – is an important consideration in the evaluation of the security of many ciphers against *differential attacks*. This paper shows that the satisfiability of an arbitrary set of DEA – which has so far been assumed *hard* for large n – is in the complexity class P. We also design an efficient algorithm to obtain all solutions to an arbitrary system of DEA with running time linear in the number of solutions. Our second contribution is solving DEA in an *adaptive query model* where an equation is formed by a query (α, β) and oracle output γ. The challenge is to optimize the number of queries to solve $(x+y) \oplus ((x \oplus \alpha) + (y \oplus \beta)) = \gamma$. Our algorithm solves this equation with only 3 queries in the worst case. Another algorithm solves the equation $(x+y) \oplus (x+(y \oplus \beta)) = \gamma$ with $(n-t-1)$ queries in the worst case (t is the position of the least significant '1' of x), and thus, outperforms the previous best known algorithm by Muller – presented at FSE '04 – which required $3(n-1)$ queries. Most importantly, we show that the upper bounds, for our algorithms, on the number of queries match worst case lower bounds. This, essentially, closes further research in this direction as our lower bounds are *optimal*.

We used our results to cryptanalyze a recently proposed cipher Helix, which was a candidate for consideration in the 802.11i standard. We are successful in reducing the data complexity of a DC attack on the cipher by a factor of 3 in the worst case (a factor of 46.5 in the best case).

Keywords. Differential Cryptanalysis, Helix cipher, P, NP.

*This work was supported in part by the Concerted Research Action (GOA) Mefisto 2000/06 and Ambiorix 2005/11 of the Flemish Government and in part by the European Commission through the IST Programme under Contract IST-2002-507932 ECRYPT. The information in this document reflects only the authors' views, is provided as is and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

Contents

Page

1 Introduction **3**
- 1.1 Notation and Model of Computation . . . 4

2 Solving an Arbitrary System of DEA **5**
- 2.1 Computing the *Character Set* and the *Useful Set* . . . 5
- 2.2 Precomputation . . . 6
- 2.3 Computation of Parameters G_i, $S_{i,0}$ and $S_{i,1}$ from the *Useful Set* $\tilde{A}$ . . . 6
- 2.4 Satisfiability of DEA is in P . . . 7
- 2.5 Computing All the Solutions to a System of DEA . . . 8

3 Solving DEA in the Adaptive Query Model **8**
- 3.1 The Power of the Adversary . . . 9
- 3.2 The Task . . . 9
- 3.3 Number of Solutions . . . 10
- 3.4 Worst Case Lower Bounds on the Number of Queries . . . 11
- 3.5 Optimal Algorithms . . . 13

4 Cryptographic Applications: Attacking the Helix Cipher **15**

5 Conclusion and Further Research **17**

6 Acknowledgments **17**

A Appendix **19**
- A.1 Proofs of Lemma 1 and 2 . . . 19
- A.2 Proof of Lemma 3 . . . 19
- A.3 Construction of M from the L_i's . . . 20
- A.4 Proof of Lower Bound for (3) . . . 20

1 Introduction

Addition modulo 2^n. Mixing *addition modulo* 2^n $(+)$ with other Boolean operations such as *exclusive-or* $(\oplus)$, *or* $(\vee)$ and/or *and* $(\wedge)$ is extensively used in symmetric cryptography. The main motivation for including *addition mod* 2^n in cryptographic primitives is that it is a nonlinear transformation over GF(2) and the operation is extremely fast on all present day architectures. Nonlinear transformations are of paramount importance in the design of ciphers as they make functions hard to invert. Helix [FW+03], IDEA [LM+91], Mars [BC+98], RC6 [RR+99], and Twofish [SK+99] which mix modular *addition* with *exclusive-or* are a few examples of the application of *addition*. Very recently Klimov and Shamir also used an update function for internal state, known as a T-function, where *addition* is mixed with *multiplication* and *or* in a certain fashion to achieve many useful properties of a secure stream cipher [KS04, KS03].

Keeping with the trend of widespread use of *addition* in symmetric ciphers, there is a large body of literature that studies equations involving *addition* from many different angles. Staffelbach and Meier investigated the probability distribution of the carry for integer addition [SM90]. Wallén explained the linear approximations of modular addition [Wal03]. Lipmaa and Moriai [LM02] investigated the equation $(x+y) \oplus ((x \oplus \alpha) + (y \oplus \beta)) = \gamma$, where α, β are the input differences and γ is the output difference, to compute many differential properties. The dual of the above equation $(x \oplus y) + ((x+\alpha) \oplus (y+\beta)) = \gamma$ was investigated for differential properties by Lipmaa *et al.* [LW+04].

Differential Cryptanalysis (DC). Differential Cryptanalysis, introduced by Biham and Shamir [BS91], is one of the most powerful attacks against symmetric ciphers. There are broadly two lines of attacks based on DC. One is guessing input or output differences of a cipher with nontrivial probability. In a cipher that is secure against DC, input and output differences should behave 'pseudorandomly', so that none of them can be guessed from any known values with a nontrivial probability. This line of attack usually results in distinguishing attacks [YB+04]. A second line of attack is much stronger but more difficult to implement than the other. It recovers secret information from known input and output differences, akin to the algebraic attacks [Mul04]. Note that this second line of attack implies the first but the converse is not true. Therefore, provable security against DC – introduced by Lai *et al.* [LM+91] and first implemented by Nyberg and Knudsen [NK91] – remained a key factor in the evaluation of the security of a cipher. However, security of many complex modern ciphers against DC is hard to evaluate because of lack of theory to evaluate the security of its components. Our target is to mount a second line of attack (i.e., to recover secret information) on the much used symmetric cipher component *addition modulo* 2^n.

Results of the Paper. There are two basic *addition* equations under DC where differences of inputs and outputs are expressed as *exclusive-or*.

$$(x+y) \oplus (x + (y \oplus \beta)) = \gamma\,, \tag{1}$$

$$(x+y) \oplus ((x \oplus \alpha) + (y \oplus \beta)) = \gamma\,. \tag{2}$$

These equations are named *differential equations addition* (DEA). While engaged in cryptanalysis of MD5, Berson noted in 1992 that, for large n, it is *hard* to analyze modular addition when differences are expressed as XOR [Ber92]. This may have motivated the use of *addition* in conjunction with XOR in many symmetric ciphers to increase resistance against DC.

In this paper we show that the satisfiability of a randomly generated set of DEA is in the complexity class P. In other words, a *Turing machine* can show in $\mathcal{O}(n^k)$ time whether there exists a solution to an arbitrary set of DEA (n denotes the bit-length of x, y and $k > 0$ is an integer-valued constant computed from the degree of the polynomial (in n) by which the number of equations to be solved is bounded above). This result, on one hand, gives deeper insight into the behavior of addition under DC. On the other hand this leaves a cautionary note for the cryptographers to be more careful about using addition in the design. Outside cryptography, satisfiability of a system

of equations has a natural appeal to many areas such as computational complexity, combinatorics, circuit optimization and computer algebra (remember the most famous NP-Complete satisfiability problem: Boolean formula satisfiability [CL$^+$90]). For example, if a large system of DEA is NOT satisfiable then the whole circuit representing the system of DEA can be safely removed to optimize the circuit complexity. Going beyond the satisfiability problem, we also give an efficient algorithm to compute all the solutions to a randomly generated system of DEA with running time linear in the number of solutions. Another subtle but a noteworthy aspect of our work is the departure from the traditional technique for solving multivariate polynomial equations over GF(2) [AL95]. We heavily benefit from certain properties of DEA and solve such systems combinatorially.

Next, we extend our work to solve DEA in a crypto-friendly *adaptive query model*. The aim is to *minimize* the search space for the secret (x, y) using a *minimum* number of adaptive queries (α, β). Such an optimization problem – typically used to reduce data complexity of chosen plaintext attacks – for (1) has already been tackled by Muller [Mul04]. But an optimal solution has been elusive until now. We achieve optimal solutions for both of the equations. We show that a worst case lower bound on the number of queries $(0, \beta)$ to solve (1) is $(n-t-1)$ where $(n-t) > 1$ with t being the bit-position of the least significant '1' of x. A worst case lower bound on the number of queries (α, β) to solve (2) is 3 for $n > 2$. Most importantly, for solving the above equations we also design algorithms whose upper bounds on the number of queries match worst case lower bounds. Note that our algorithm outperforms the previous best known algorithm by Muller to solve (1) – presented at FSE '04 – which required $3(n-1)$ queries [Mul04]. Over and above, our results essentially close further investigation in this particular direction as the equations are solved with an *optimal* number of queries in the worst case. It is particularly interesting to note that, for (2), although the number of all queries grows exponentially with the input size n, an optimal lower bound to solve (2) is 3 for all $n > 2$, i.e., constant asymptotically.

Our results on *modular addition* have the potential to be used either directly in the cryptanalysis of ciphers that include this special component or to facilitate cryptanalysis of several modern ciphers (e.g., mixing addition with multiplication). We show that, with a maximum of only 3 adaptively chosen queries, the search space of the secret of modular addition against DC can be reduced from 2^{2n} to only 4 for all $n \geq 1$. We used our results to cryptanalyze a recently proposed cipher Helix [FW$^+$03] which was a candidate for consideration in the 802.11i standard. We are successful in reducing the data complexity of a DC attack on the cipher by a factor of 3 in the worst case (a factor of 46.5 in the best case) [Mul04]. In addition, using our algorithm to solve DEA, as discussed above we are able to compute all the *differential properties of addition* by investigating a single equation (note that the *differential properties of addition* have been independently found by Lipmaa and Moriai using a different technique [LM02]).

1.1 Notation and Model of Computation

The purpose of the paper is to solve (1) and (2) for (x, y) using triples (α, β, γ) where $x, y, \alpha, \beta, \gamma \in \mathbb{Z}_2^n$. The ith bit of an n-bit integer l is denoted by l_i (l_0 denotes the least significant bit or the 0th bit of l). The operation *addition modulo* 2^n over $\mathbb{Z}_{2^n}$ can be viewed as a binary operation over $\mathbb{Z}_2^n$ (we denote this operation by '+') using the bijection that maps $(l_{n-1}, \cdots, l_0) \in \mathbb{Z}_2^n$ to $l_{n-1}2^{n-1} + \cdots + l_0 2^0 \in \mathbb{Z}_{2^n}$. Therefore '+' is a function '+': $\mathbb{Z}_2^n \times \mathbb{Z}_2^n \rightarrow \mathbb{Z}_2^n$. The symbols '$\oplus$' and '$\wedge$' denote the operations *bit-wise exclusive-or* and *bit-wise and* of two n-bit integers respectively. We will denote $a \wedge b$ by ab. Throughout the paper, $[p, q]$ denotes a set containing all integers between the integers p and q including both of them. Unless otherwise stated, n denotes a positive integer. The size of a set S is denoted by $|S|$.

The algorithms, described in this paper, can be implemented on a generic one-processor *Random Access Machine* (RAM) (i.e., instructions are executed sequentially) whose memory is composed of an unbounded sequence of registers each capable of containing an integer. Typically, *RAM* instructions consist of simple arithmetic operations (addition and bitwise XOR in our case), storing,

addressing (direct and indirect) and branching, each of which is a constant time operation. However, the choice of *RAM* instructions is relatively less important because algorithms based on two reasonable sets of instructions will have the same asymptotic complexity. A detailed analysis of *RAM* can be found in [AH$^+$, FB]. It can be shown that a polynomial-time solvable problem on a *RAM* is also polynomial-time solvable on a *Turing Machine* and vice versa.

2 Solving an Arbitrary System of DEA

Our aim is to solve for $(x,\, y)$ from the following set of differential equations of addition over $\mathbb{Z}_2^n$,

$$(x+y) \oplus ((x \oplus \alpha[k]) + (y \oplus \beta[k])) = \gamma[k]\,, \quad k = 1,\, 2 \cdots m\,. \tag{3}$$

We observe that the ith bit of $\gamma[k]$ can be written as

$$\gamma[k]_i = x_i \oplus y_i \oplus c_i \oplus \tilde{x_i} \oplus \tilde{y_i} \oplus \tilde{c_i}, \quad i \in [0,\, n-1]\,, \tag{4}$$

where $\tilde{x_i} = x_i \oplus \alpha[k]_i$ and $\tilde{y_i} = y_i \oplus \beta[k]_i$ and c_i, $\tilde{c_i}$ are computed from the following recursion,

$$c_0 = \tilde{c_0} = 0,\, c_{j+1} = x_j y_j \oplus x_j c_j \oplus y_j c_j,\, \tilde{c}_{j+1} = \tilde{x_j}\tilde{y_j} \oplus \tilde{x_j}\tilde{c_j} \oplus \tilde{y_j}\tilde{c_j}, \quad j \in [0,\, n-2]\,. \tag{5}$$

Thus we see that $\gamma[k]_i$ is a function of the least $(i+1)$ bits of x, y, $\alpha[k]$ and $\beta[k]$. More formally,

$$\gamma[k]_i = F_i(x_0, \cdots,\, x_i,\, y_0, \cdots,\, y_i,\, \alpha[k]_0, \cdots,\, \alpha[k]_i,\, \beta[k]_0, \cdots,\, \beta[k]_i)\,. \tag{6}$$

Therefore, from a system of m differential equations of addition, a total of mn multivariate polynomial equations over GF(2) can be formed by ranging $(k,\, i)$ through all values in (6).

Plenty of research has been undertaken to design efficient ways to solve randomly generated multivariate polynomial equations. The classical Buchberger's Algorithm for generating Gröbner bases [AL95] and its variants [Fau99] are some of them. This problem is NP-complete (NPC) over GF(2). Many other techniques such as relinearization [KS99] have been proposed to solve a special case of overdefined systems of multivariate polynomial equations. Note that, in our case, the number of unknowns and equations are $2n$ and mn respectively (if $m > 2$ then the system of equations is overdefined). However, taking full advantage of the specific nature of the *differential equations of addition*, we shall use a combinatorial technique to prove that, although the satisfiability of an arbitrary multivariate polynomial equation over GF(2) is NP-complete, this special cryptographically important subclass of equations is in the complexity class P (see [CL$^+$90], [HM$^+$04] for definitions of NP, P, NPC). Finally, we also derive all the solutions to a system of such equations.

2.1 Computing the *Character Set* and the *Useful Set*

From (3) we construct $A = \{(\alpha[k],\, \beta[k],\, \gamma[k])\,|\,k \in [1,\, m]\}$ assuming $(\alpha[k],\, \beta[k],\, \gamma[k])$'s are all distinct.[1] We call A the *character set*. Our first step is to transform the system of equations defined in (3) into a new set of equations over $\mathbb{Z}_2^n$ as defined below,

$$(x+y) \oplus ((x \oplus \alpha[k]) + (y \oplus \beta[k])) \oplus \alpha[k] \oplus \beta[k] = \tilde{\gamma}[k]\,, \quad k = 1,\, 2 \cdots m\,; \tag{7}$$

where $\tilde{\gamma}[k] = \gamma[k] \oplus \alpha[k] \oplus \beta[k]$. Now, we construct $\tilde{A}$,

$$\tilde{A} = \{(\alpha,\, \beta,\, \tilde{\gamma} = \alpha \oplus \beta \oplus \gamma)\,|\,(\alpha,\, \beta,\, \gamma) \in A\}\,. \tag{8}$$

We call $\tilde{A}$ the *useful set*. Let all the solutions for (3) and (7) be contained in the sets A-satisfiable and $\tilde{A}$-consistent respectively. It is trivial to show that

$$A\text{-satisfiable} = \tilde{A}\text{-consistent}\,. \tag{9}$$

Our aim is to compute $\tilde{A}$-consistent from $\tilde{A}$.

[1] This can be obtained by taking one of the identical equations in (3).

2.2 Precomputation

Take an arbitrary element $(\alpha,\ \beta,\ \tilde{\gamma}) \in \tilde{A}$ $(n > 1)$. Observe that $\tilde{\gamma}_{i+1}$ can be computed using $x_i,\ y_i,\ c_i,\ \alpha_i,\ \beta_i, \tilde{\gamma}_i,\ \forall\, i \in [0,\ n-2]$, from the following three equations

$$\tilde{\gamma}_{i+1} = c_{i+1} \oplus \tilde{c}_{i+1},\ c_{i+1} = x_i y_i \oplus x_i c_i \oplus y_i c_i,\ \tilde{c}_{i+1} = \tilde{x}_i \tilde{y}_i \oplus \tilde{x}_i \tilde{c}_i \oplus \tilde{y}_i \tilde{c}_i$$

where c_i is the carry at the ith position of $(x + y)$, $\tilde{x}_i = x_i \oplus \alpha_i$, $\tilde{y}_i = y_i \oplus \beta_i$ and $\tilde{c}_i = c_i \oplus \tilde{\gamma}_i$. Table 1 lists the values of $\tilde{\gamma}_{i+1}$ as computed from all values of $x_i,\ y_i,\ c_i,\ \alpha_i,\ \beta_i,\ \tilde{\gamma}_i$.

Table 1: The values of $\tilde{\gamma}_{i+1}$ corresponding to the values of $x_i,\ y_i,\ c_i,\ \alpha_i,\ \beta_i,\ \tilde{\gamma}_i$. A row and a column are denoted by R(l) and Col(k)

(x_i, y_i, c_i)	$(\alpha_i,\ \beta_i, \tilde{\gamma}_i)$								
	(0,0,0)	(0,0,1)	(0,1,0)	(0,1,1)	(1,0,0)	(1,0,1)	(1,1,0)	(1,1,1)	R(0)
(0,0,0)	0	0	0	1	0	1	1	1	R(1)
(1,1,1)									
(0,0,1)	0	0	1	0	1	0	1	1	R(2)
(1,1,0)									
(0,1,0)	0	1	0	0	1	1	0	1	R(3)
(1,0,1)									
(1,0,0)	0	1	1	1	0	0	0	1	R(4)
(0,1,1)									
Col(0)	Col(1)	Col(2)	Col(3)	Col(4)	Col(5)	Col(6)	Col(7)	Col(8)	

2.3 Computation of Parameters G_i, $S_{i,0}$ and $S_{i,1}$ from the *Useful Set* $\tilde{A}$

We now determine an important quantity, denoted by G_i, for a nonempty *useful set* $\tilde{A}$. In G_i, we store the ith and $(i+1)$th bits of $\tilde{\gamma}$ and the ith bit of α and β for all $(\alpha,\ \beta,\ \tilde{\gamma}) \in \tilde{A}$. We call G_i the ith core of the *useful set* $\tilde{A}$. More formally (suppose $n > 1$),

$$G_i = \{(\alpha_i,\ \beta_i,\ \tilde{\gamma}_i,\ \tilde{\gamma}_{i+1}) \,|\, (\alpha,\ \beta,\ \tilde{\gamma}) \in \tilde{A}\}, \quad i \in [0,\ n-2]\,. \tag{10}$$

In the subsequent discussion we will often use the expression "$G_i \Rightarrow (x_i,\ y_i,\ c_i)$". Let $|G_i| = g$. Take an element $(\alpha_i,\ \beta_i,\ \tilde{\gamma}_i,\ \tilde{\gamma}_{i+1}) \in G_i$. In Table 1, find the row(s) of the fourth coordinate $\tilde{\gamma}_{i+1}$ in the column specified by the first three coordinates $(\alpha_i,\ \beta_i,\ \tilde{\gamma}_i)$ in R(0) and put them in set F_{i1}. Find $F_{i1}, \cdots F_{ig}$ for all g elements of G_i. Let $F_i = \bigcap_j F_{ij}$ and R(x)$\in F_i$. If $(x_i,\ y_i,\ c_i)$ is in Col(0)$\times$R(x) then we say $G_i \Rightarrow (x_i,\ y_i,\ c_i)$. If $F_i = \phi$ then no such $(x_i,\ y_i,\ c_i)$ exists. We compute $S_{i,j} = \{(x_i,\ y_i) \,|\, G_i \Rightarrow (x_i,\ y_i,\ c_i = j)\}$.

Example. $(G_i, S_{i,0}, S_{i,1})$ Let $n = 3$ and the *useful set* $\tilde{A} = \{((0,\ 1,\ 0), (1,\ 0,\ 1)\ , (0,\ \ 0,\ 0)),$ $((0, 0,\ 0),\ (1,\ 1,\ \ 1),\ (1, 0,\ 0)),\ ((0,\ \ 0,\ 1),\ \ (0,\ 1,\ \ 1),\ \ (1,\ 1,\ 0))\}$. Therefore, $G_0 = \{(0,\ 1,\ 0,\ 0),$ $(1,\ 1,\ 0,\ 1)\}$, $G_1 = \{(1,\ 0,\ 0,\ 0),\ \ (0,\ \ 1,\ \ 0,\ 1),\ \ (0,\ 1,\ 1,\ 1)\}$ (see (10)). Now, from Table 1, $F_{01} =$ $\{R(1),\ R(3)\}$, $F_{02} = \{R(1),\ R(2)\}$, $F_{11} = \{R(1),\ R(4)\}$, $F_{12} = \{R(2),\ R(4)\}$, $F_{13} = \{R(1),\ R(4)\}$. Therefore, $F_0 = F_{01} \cap F_{02} = \{R(1)\}$ and $F_1 = F_{11} \cap F_{12} \cap F_{13} = \{R(4)\}$. Now $G_0 \Rightarrow (0,\ 0,\ 0)$, $G_0 \Rightarrow (1,\ 1,\ 1)$ as $(0,\ 0,\ 0)$, $(1,\ 1,\ 1)$ are in Col(0)$\times$R(1). Similarly, $G_1 \Rightarrow (1,\ 0,\ 0)$, $G_1 \Rightarrow (0,\ 1,\ 1)$. Thus, $S_{0,0} = \{(0,\ 0)\}$, $S_{0,1} = \{(1,\ 1)\}$, $S_{1,0} = \{(1,\ 0)\}$, $S_{1,1} = \{(0,\ 1)\}$. □

We assume $m = |\tilde{A}| = \mathcal{O}(n^l)$ for some nonnegative integer l. Then the time and memory to compute all G_i's and $S_{i,j}$'s are $\mathcal{O}(n^k)$ each because the size of Table 1 and $|G_i|$ are $\mathcal{O}(1)$ each $(k = l + 1)$. Now, we show a relation between $S_{i,0}$ and $S_{i,1}$ that will be used to obtain several results.

Proposition 1 *For all nonempty useful set $\tilde{A}$ and all $n > 1$, $|S_{i,0}| = |S_{i,1}| \ \forall\, i \in [0,\, n-2]$.*

We set,

$$|S_{i,0}| = |S_{i,1}| = S_i \quad \forall\, i \in [0,\, n-2]. \tag{11}$$

2.4 Satisfiability of DEA is in P

In this section, we deal with a decision problem: does there exist a solution for an arbitrary set of differential equations of addition, i.e., is a system of DEA satisfiable?

We have already seen how to compute the *character set* A, the *useful set* $\tilde{A}$, the core G_i's and S_i's from a system of DEA in $\mathcal{O}(n^k)$ (see Sect. 2.3). Now, we prove an important theorem that characterizes the membership in $\tilde{A}$-consistent which is, in fact, the solution set.

Theorem 1 *Let the useful set $\tilde{A} \neq \phi$ and $n > 1$. The following two statements are equivalent.*
1. $(x,\, y) \in \mathbb{Z}_2^n \times \mathbb{Z}_2^n$ is such that $G_i \Rightarrow (x_i,\, y_i,\, c_i),\ \forall\, i \in [0,\, n-2]$.
2. $(x,\, y) \in \tilde{A}$-consistent.

Proof. From the construction of G_i, it can be shown that $1 \Leftrightarrow 2$. □

Armed with the above theorem, we now formulate $|\tilde{A}$-consistent$|$ (i.e., the number of solutions) in the following proposition which will later answer our satisfiability question.

Proposition 2 *Let the useful set $\tilde{A} \neq \phi$ and S denote $|\tilde{A}$-consistent$|$. Then,*

$$S = \begin{cases} 0 & \text{if } \tilde{\gamma}_0 = 1 \text{ for some } (\alpha,\, \beta,\, \tilde{\gamma}) \in \tilde{A}, \\ 4 \cdot \prod_{i=0}^{n-2} S_i & \text{if } \tilde{\gamma}_0 = 0,\ \forall (\alpha,\, \beta,\, \tilde{\gamma}) \in \tilde{A} \text{ and } n > 1, \\ 4 & \text{if } \tilde{\gamma}_0 = 0,\ \forall (\alpha,\, \beta,\, \tilde{\gamma}) \in \tilde{A} \text{ and } n = 1. \end{cases}$$

The S_i's are defined in (11).

Proof. We prove this result by considering all cases individually.

Case 1 If $\tilde{\gamma}_0 = 1$ for some $(\alpha,\, \beta,\, \tilde{\gamma}) \in \tilde{A}$ then $c_0 = 1$ which is impossible.

Case 2 If $\tilde{\gamma}_0 = 0,\ \forall (\alpha,\, \beta,\, \tilde{\gamma}) \in \tilde{A}$ and $n > 1$. From Theorem 1 of Sect. 2.5, S is the number of solutions $(x,\, y) \in \mathbb{Z}_2^n \times \mathbb{Z}_2^n$ such that $G_i \Rightarrow (x_i,\, y_i,\, c_i), \forall i \in [0,\, n-2]$. Let M_k denote the number of solutions for $((x_k,\, \cdots, x_0),\, (y_k,\, \cdots, y_0))$ such that $G_i \Rightarrow (x_i,\, y_i,\, c_i), \forall i \in [0,\, k]$ where $k \in [0,\, n-2]$. Note that G_k depends only on the least $(k+1)$ bits of x, y, α, β. We consider two subcases.
Case 2(a): $n > 2$. We determine $|\tilde{A}$-consistent$|$ recursively. Let $M_l = M_{l,0} + M_{l,1}$ such that $M_{l,0}$ solutions produce $c_{l+1} = 0$ and $M_{l,1}$ solutions produce $c_{l+1} = 1$. Therefore, $\forall\, l \in [0,\, n-3]$

$$\begin{aligned} M_{l+1} &= M_{l,0} \cdot |S_{l+1,0}| + M_{l,1} \cdot |S_{l+1,1}| \\ &= S_{l+1} \cdot M_l\,. \end{aligned} \tag{12}$$

as $|S_{i,0}| = |S_{i,1}| = S_i,\ \forall\, i \in [0,\, n-2]$ (see Proposition 1). It is easy to show (a proof is by contradiction) that M_{l+1}, so calculated, gives the number of solutions for $((x_{l+1},\, \cdots, x_0), (y_{l+1},\, \cdots, y_0))$ such that $G_i \Rightarrow (x_i,\, y_i,\, c_i), \forall i \in [0,\, l+1]$. From (12),

$$M_{n-2} = \prod_{i=0}^{n-2} S_i \tag{13}$$

as $M_0 = S_0$. Note that, for all $(\alpha,\, \beta,\, \tilde{\gamma}) \in \tilde{A}$, $\tilde{\gamma}$ is independent of $(x_{n-1},\, y_{n-1})$. Therefore,

$$S = 4 \cdot M_{n-2} = 4 \cdot \prod_{i=0}^{n-2} S_i \quad \text{if } n > 2\,. \tag{14}$$

Case 2(b): $n = 2$. It is easy to show that $S = 4 \cdot S_0$ if $n = 2$.

Case 3 If $\tilde{\gamma}_0 = 0$, $\forall(\alpha, \beta, \tilde{\gamma}) \in \tilde{A}$ and $n = 1$. It is trivial to show that $S = 4$ if $n = 1$ since for all $(\alpha, \beta, \tilde{\gamma}) \in \tilde{A}$, $\tilde{\gamma}$ is independent of (x_{n-1}, y_{n-1}). □

Using Proposition 2, we now answer the question of satisfiability of DEA in the following claim.

Claim. *(i) If $\tilde{\gamma}_0 = 1$ for some $(\alpha, \beta, \tilde{\gamma}) \in \tilde{A}$, then the set of DEA is NOT satisfiable. (ii) If $\tilde{\gamma}_0 = 0$, $\forall(\alpha, \beta, \tilde{\gamma}) \in \tilde{A}$ and $n > 1$, then the set of DEA is satisfiable if and only if $S_i \neq 0$ $\forall i \in [0, n-2]$. (iii) If $\tilde{\gamma}_0 = 0$, $\forall(\alpha, \beta, \tilde{\gamma}) \in \tilde{A}$ and $n = 1$, then the set of DEA is satisfiable.*

Verification of *(i)*, *(ii)* and *(iii)* take time $\mathcal{O}(1)$, $\Theta(n)$ and $\mathcal{O}(1)$ respectively. Therefore, the overall time to decide whether a system of DEA is satisfiable is $\mathcal{O}(n^k) + \mathcal{O}(1) + \Theta(n) + \mathcal{O}(1) = \mathcal{O}(n^k)$. Thus the satisfiability of DEA is in P.

2.5 Computing All the Solutions to a System of DEA

Now the only part left unanswered is how to actually compute $\tilde{A}$-consistent, i.e., to extract all the solutions of a system of DEA which is satisfiable. Note, if $n = 1$ then $\tilde{A}$-consistent comprises all 4 values of (x, y). The G_i's can be computed from the *useful set* $\tilde{A}$ in $\mathcal{O}(n^k)$ (see Sect. 2.3). Now we compute an intermediate parameter $L_i = \{(x_i, y_i, c_i) \,|\, G_i \Rightarrow (x_i, y_i, c_i)\}$ for all $i \in [0, n-2]$ (note that L_i is different from F_i which has been computed in Sect. 2.3). Computation of the L_i's takes time and memory each $\Theta(n)$. We call L_i the ith bit solution. Algorithm 1 computes $\tilde{A}$-consistent from the L_i's ($n > 1$).

Algorithm 1 Compute all the solutions to a system of satisfiable DEA

Require: L_i, $\forall i \in [0, n-2]$
Ensure: $\tilde{A}$-consistent
1: Compute $M = \{((x_{n-1}, x_{n-2}, \cdots, x_0), (y_{n-1}, y_{n-2}, \cdots, y_0)) \,|\, (x_{n-1}, y_{n-1}) \in \mathbb{Z}_2^2,$
$(x_i, y_i, c_i) \in L_i,\ i \in [0, n-2],\ c_0 = 0, c_{i+1} = x_i y_i \oplus x_i c_i \oplus y_i c_i\}$.
2: Return (M).

The idea of the algorithm is to collect in M all $(x, y) \in \mathbb{Z}_2^n \times \mathbb{Z}_2^n$ such that $G_i \Rightarrow (x_i, y_i, c_i)$, $\forall i \in [0, n-2]$. Theorem 1 is the heart of the argument to prove that M is essentially $\tilde{A}$-consistent.

Time and Memory. Algorithm 1 takes time $\Theta(S)$ and memory $\Theta(n \cdot S)$ where S is the number of solutions (an explicit construction of M from the L_i's and its complexity analysis are shown in Appendix A.3).

3 Solving DEA in the Adaptive Query Model

In this section, we solve the equations

$$(x + y) \oplus ((x \oplus \alpha) + (y \oplus \beta)) = \gamma, \tag{15}$$

$$(x + y) \oplus (x + (y \oplus \beta)) = \gamma \tag{16}$$

separately in an *adaptive query model*. Solving (15) in *adaptive query model* means solving a set of 2^{2n} equations generated by ranging (α, β) with the corresponding γ. The number of solutions satisfying 2^{2n} equations is less than that of any subset of the equations. Therefore, solving these 2^{2n} equations reduces the search space of the secret (x, y) to the minimum. This fact is the major motivation for dealing with this problem. The task of a computationally unbounded adversary is to select a subset A of all equations such that the solutions to the chosen subset A are the same as that of the entire 2^{2n} equations. The target of the adversary is to minimize $|A|$. A similar optimization

problem can be asked of (16) where the number of equations is 2^n. Such an optimization problem for (16) has already been tackled by Muller [Mul04] in cryptanalysis of the Helix cipher but an optimal solution has still been elusive. We reach optimal solutions for both equations.

3.1 The Power of the Adversary

The power of an adversary that solves (15) is defined as follows.

1. An adversary has unrestricted computational power and an infinite amount of memory.
2. An adversary can *only* submit queries $(\alpha, \beta) \in \mathbb{Z}_2^n \times \mathbb{Z}_2^n$ to an honest oracle[2] which computes γ using fixed unknown $(x, y) \in \mathbb{Z}_2^n \times \mathbb{Z}_2^n$ in (15) and returns the value to the adversary. We will often refer to that fixed (x, y) as the *seed* of the oracle.

Such an oracle with seed (x, y) is viewed as a mapping $O_{xy} : \mathbb{Z}_2^n \times \mathbb{Z}_2^n \to \mathbb{Z}_2^n$ and defined by

$$O_{xy} = \{(\alpha, \beta, \gamma) \,|\, (\alpha, \beta) \in \mathbb{Z}_2^n \times \mathbb{Z}_2^n, \gamma = (x + y) \oplus ((x \oplus \alpha) + (y \oplus \beta))\}\,. \tag{17}$$

An adversarial model, similar to the one described above for (15), can be constructed for (16) by setting $(\alpha, \beta) \in \{0\}^n \times \mathbb{Z}_2^n$ and the mapping $O_{xy} : \{0\}^n \times \mathbb{Z}_2^n \to \mathbb{Z}_2^n$.

The model described above represents a practical adaptively chosen message attack scenario where the adversary makes adaptive queries to an oracle. Based on the replies from the oracle, the adversary computes one or more unknown parameters.

3.2 The Task

O_{xy}, defined in (17), generates a family of mappings $\mathcal{F} = \{O_{xy} \,|\, (x, y) \in \mathbb{Z}_2^n \times \mathbb{Z}_2^n\}$. Note that, if $D \in \mathcal{F}$ then $|D| = 2^{2n}$ for (15). Therefore, $D \in \mathcal{F}$ is the *character set* with the number of equations $m = 2^{2n}$ (see Sect. 2.1). Our aim is to find all (x, y) satisfying these 2^{2n} equations, i.e., to compute D-satisfiable from a subset of the *character set* D. If we deal with (16) then $|D| = 2^n$.

An Equivalent Task. From the *character set* D one can compute the *useful set* $\tilde{D}$ using (8). Therefore, the task is equivalent to the determination of $\tilde{D}$-consistent from a subset of the *useful set* $\tilde{D}$. We call D and $\tilde{D}$ the *total character set* and the *total useful set* as their sizes are maximal and they are generated from a satisfiable set 2^{2n} DEA (because we assumed the oracle to be honest). Note that there is a bijection between D and $\tilde{D}$.

Adjusting the Oracle Output. If the oracle outputs γ on query (α, β), we shall consider the oracle output to be $\tilde{\gamma} = \alpha \oplus \beta \oplus \gamma$ for the sake of simplicity in the subsequent discussions.

Rules of the Game. Now we lay down the rules followed by the adversary to determine the set $\tilde{D}$-consistent that, in turn, gives the essence of the whole problem.

1. The adversary starts with no information about x and y except their size n.
2. The adversary settles on a strategy (i.e., a deterministic algorithm) which is publicly known. Using the strategy, the adversary computes queries adaptively, i.e., based on the previous queries and the corresponding oracle outputs the next query is determined.
3. The game stops the moment the adversary constructs $\tilde{D}$-consistent. The adversary fails if she is unable to compute $\tilde{D}$-consistent for some $(x, y) \in \mathbb{Z}_2^n \times \mathbb{Z}_2^n$.

[2] An honest oracle correctly computes γ and returns it to the adversary.

We search for an algorithm that determines $\tilde{D}$-consistent for all $(x,\ y) \in \mathbb{Z}_2^n \times \mathbb{Z}_2^n$. Furthermore, there is an additional requirement that, in the worst case of $(x,\ y)$, the number of queries required by the algorithm is the minimum . We shall elaborate on the meaning of *worst case* in Sect. 3.4 which focuses on worst case lower bounds on the number of queries.

3.3 Number of Solutions

In this section we are interested to determine the number of solutions of (15) and (16) in the adaptive query model. We have already developed a framework where the set of all solutions in the adaptive query model is denoted by $\tilde{D}$-consistent where $\tilde{D}$ is the *total useful set*. Therefore, formally, our effort will be directed to formulate $|\tilde{D}\text{-consistent}|$. We will see in Theorem 5 that, for (16), $|\tilde{D}\text{-consistent}|$ depends on the least significant '1' of x. However, for (15), $|\tilde{D}\text{-consistent}| = 4$, $\forall (x,\ y) \in \mathbb{Z}_2^n \times \mathbb{Z}_2^n$. We shall use these results in Theorem 5 and 6 of Sect. 3.4, to obtain lower bounds on the number of queries to compute $\tilde{D}$-consistent and in Sect. 3.5, to prove the correctness of our optimal algorithms.

Theorem 2 *Let the position of the least significant '1' of x in the equation*

$$(x + y) \oplus (x + (y \oplus \beta)) = \gamma$$

be t and $x,\ y,\ \beta,\ \gamma \in \mathbb{Z}_2^n$. Let the total useful set $\tilde{D}$ *be given. Then $|\tilde{D}$-consistent$|$ is*
(i) 2^{t+3} if $n - 2 \geq t \geq 0$,
(ii) 2^{n+1} otherwise.

Proof.
(i) $n - 2 \geq t \geq 0$. Using the procedure described in Sect. 2.3, we construct the ith core G_i, $\forall\, i \in [0,\ n-2]$, from the *total useful set* $\tilde{D}$. We derive that

$$G_i = \{(0,\ 0,\ 0,\ a_i),\ (0,\ 1,\ 0,\ b_i)\}\,, \qquad \forall i \in [0,\ t] \tag{18}$$

$$G_i = \{(0,\ 0,\ 0,\ c_i),\ (0,\ 0,\ 1,\ d_i)\,(0,\ 1,\ 0,\ e_i),\ (0,\ 1,\ 1,\ f_i)\},\ \ \forall i \in [t+1,\ n-2]\,. \tag{19}$$

In the above equations, $a_i,\ b_i,\ c_i,\ d_i,\ e_i,\ f_i \in [0,\ 1]$. Note that only (18) is relevant if $t = n - 2$. The fact that we are able to extract only the first three coordinates of the elements of G_i, $\forall i \in [0,\ n-2]$, can be proved using the following two auxiliary lemmas, the proofs of which are given in Appendix A.1.

Lemma 1 *For all $(0,\ \beta,\ \tilde{\gamma}) \in \tilde{D}$, $\tilde{\gamma}_i = 0,\ \forall\, i \in [0,\ t]$.*

Lemma 2 *For all $i \in [t+1,\ n-1]$ there exists $(0,\ \beta,\ \tilde{\gamma}) \in \tilde{D}$ with $\tilde{\gamma}_i = 1$.*

However, only the first three coordinates of the elements of the G_i's are sufficient to determine the S_i's (S_i is defined in Sect. 2.3). It is easy to verify from Table 1 that $S_i = 2$, $\forall i \in [0,\ t]$ and $S_i = 1$, $\forall i \in [t+1,\ n-2]$. From Proposition 2

$$|\tilde{D}\text{-consistent}| = S = 4 \cdot \prod_{i=0}^{n-2} S_i = 4 \cdot \underbrace{1 \cdot 1 \cdots 1}_{(n-t-2)\text{ times}} \cdot \underbrace{2 \cdot 2 \cdots 2}_{(t+1)\text{ times}} = 2^{t+3}\,.$$

(ii) The proof is similar to the above one using Proposition 2. □

Theorem 3 *Let the* total useful set $\tilde{D}$ *be given for the equation*

$$(x + y) \oplus ((x \oplus \alpha) + (y \oplus \beta)) = \gamma$$

with $x,\ y,\ \alpha,\ \beta,\ \gamma \in \mathbb{Z}_2^n$. Then $|\tilde{D}$-consistent$|$=4.

Proof. Our approach is same as that of Theorem 2.
Case 1: $n \geq 2$**.** Corresponding to the *total useful set* $\tilde{D}$ we determine G_i, $\forall\, i \in [0,\, n-2]$.

$$G_0 = \{(0,\,0,\,0,\,a_0),\,(0,\,1,\,0,\,b_0),\,(1,\,0,\,0,\,c_0),\,(1,\,1,\,0,\,d_0)\}, \tag{20}$$

$$G_i = \{(0,\,0,\,0,\,e_i),\,(0,\,0,\,1,\,f_i),\,(0,\,1,\,0,\,g_i),\,(0,\,1,\,1,\,h_i),$$
$$(1,\,0,\,0,\,m_i),\,(1,\,0,\,1,\,n_i),\,(1,\,1,\,0,\,p_i),\,(1,\,1,\,1,\,q_i)\},\ \forall\, i \in [1,\, n-2]\,. \tag{21}$$

In the equations $a_0,\, b_0,\, c_0,\, d_0,\, e_i,\, f_i,\, g_i,\, h_i,\, m_i,\, n_i,\, p_i,\, q_i \in [0,\, 1]$. From Table 1 we see that $S_i = 1,\ \forall\, i \in [0,\, n-2]$ (see Sect. 2.3 to compute S_i from G_i). Therefore, from Proposition 2

$$|\tilde{D}\text{-consistent}| = S = 4 \cdot \prod_{i=0}^{n-2} S_i = 4 \cdot \underbrace{1 \cdot 1 \cdots 1}_{(n-1)\,\text{times}} = 4\,.$$

Case 2: $n = 1$**.** If $n = 1$ then $|\tilde{D}\text{-consistent}| = 4$. The proof is trivial using Proposition 2. □

3.4 Worst Case Lower Bounds on the Number of Queries

Our target is to design an algorithm (for (15) or (16)) which computes $\tilde{D}$-consistent for all *seeds* $(x,\, y) \in \mathbb{Z}_2^n \times \mathbb{Z}_2^n$ with adaptive queries. For such an algorithm, the number of required queries may vary with the choice of $(x,\, y)$. In this section we concentrate on a lower bound on the number of queries in the worst case of $(x,\, y)$ under the "rules of the game" stated in Sect. 3.2. The significance of the lower bound is that there exists no algorithm that requires less queries in the worst case than the obtained lower bound.

We already noticed that more queries tend to reduce the search space of the secret $(x,\, y)$. In our formal framework, if $A \subseteq B \subseteq \tilde{D}$ then $\tilde{D}$-consistent $\subseteq B$-consistent $\subseteq A$-consistent. This implies that $|\tilde{D}\text{-consistent}| \leq |B\text{-consistent}| \leq |A\text{-consistent}|$. Note that our algorithm constructs $A \subseteq \tilde{D},\ \forall (x,\, y) \in \mathbb{Z}_2^n \times \mathbb{Z}_2^n$, using the submitted queries and the corresponding outputs such that $|\tilde{D}\text{-consistent}| = |A\text{-consistent}|$. The algorithm fails if $|\tilde{D}\text{-consistent}| < |A\text{-consistent}|$, for some $(x,\, y) \in \mathbb{Z}_2^n \times \mathbb{Z}_2^n$. We will use the condition – $|A\text{-consistent}|$ cannot be *strictly* greater than $|\tilde{D}\text{-consistent}|$ – to compute a lower bound on the number of queries in the worst case. In Theorem 4 we identify a property of A, in terms of the ith core G_i, where the above condition is violated. In Theorem 5, we use this fact to obtain a lower bound.

Theorem 4 *We consider the equation*

$$(x + y) \oplus (x + (y \oplus \beta)) = \gamma\,,$$

where the position of the least significant '1' of x *is* t *with* $n - 3 \geq t \geq 0$*. Let* $\phi \subset A \subseteq \tilde{D}$ *and* $\tilde{D}$ *be the* total useful set *of the equation. Let the* $(n-3)$*th core* G_{n-3} *contain no element* $(0,\, \beta_{n-3},\, \tilde{\gamma}_{n-3},\, \tilde{\gamma}_{n-2})$ *with* $\tilde{\gamma}_{n-2} = 1$.[3] *Then* $|A\text{-consistent}| = 2^{t+3+k}$ *for some* $k > 0$*.*

Proof. If G_{n-3} contains no element $(0,\, \beta_{n-3},\, \tilde{\gamma}_{n-3},\, \tilde{\gamma}_{n-2})$ with $\tilde{\gamma}_{n-2} = 1$ then G_{n-2} contains no element $(0,\, \beta_{n-2},\, \tilde{\gamma}_{n-2},\, \tilde{\gamma}_{n-1})$ with $\tilde{\gamma}_{n-2} = 1$. Therefore, G_{n-2} is of one of the following forms,

$$G_{n-2} = \{(0,\,0,\,0,\,a)\} \quad \text{or } \{(0,\,0,\,0,\,a),\,(0,\,1,\,0,\,b)\}\,.$$

Now, from Table 1, $S_{n-2} = 2^l$ for either of the cases, where $l > 0$. Similarly, using Theorem 2, $S_i \geq 2,\ \forall\, i \in [0,\, t]$. Also $S_i \geq 1,\ \forall\, i \in [t+1,\, n-3]$ (when $n - 4 \geq t$). Therefore, from Proposition 2, $|A\text{-consistent}| = 2^{t+3+k}$ for some $k > 0$. □

In the following theorem, we partition the entire seed space $\mathbb{Z}_2^n \times \mathbb{Z}_2^n$ and compute a worst case lower bound for each partition. Note that a lower bound (say, l) for any partition shows that, for any algorithm that computes $\tilde{D}$-consistent $\forall (x,\, y) \in \mathbb{Z}_2^n \times \mathbb{Z}_2^n$, there exists at least one seed in that particular partition which requires at least l queries.

[3] This implies that there is no oracle output $\tilde{\gamma}$ with $\tilde{\gamma}_{n-2} = 1$.

Theorem 5 *A lower bound on the number of queries* $(0, \beta)$ *to solve*

$$(x + y) \oplus (x + (y \oplus \beta)) = \gamma$$

in the worst case of (x, y) *is*
(i) 0 *if* $n = 1$,
(ii) 1 *if* $x = 0$ *and* $n > 1$,
(iii) 1 *if* $n = 1 + t$ *with* $t > 0$,
(iv) $(n - t - 1)$ *if* $n - 2 \geq t \geq 0$,
where n *is the bit-length of* x, y *and* t *is the position of the least significant '1' of* x.

Proof. For *(i)*, *(ii)* and *(iii)* the lower bounds are trivial.
(iv) $n-2 \geq t \geq 0$. Note that the submission of queries, generation of the *useful set* A and the *core* G_i are the parts of an evolving process (i.e., with every submitted query $(0, \beta)$, A, G_i change). At any time, from the already submitted queries and the outputs, we can always construct $A \subseteq \tilde{D}$ and the ith core G_i, $\forall i \in [0, n-2]$ where $\tilde{D}$ is the *total useful set*. Note that A-consistent contains all the solutions of the equations derived from the already submitted queries. We first divide the case into four disjoint subcases.
Case 1: $n - 5 \geq t \geq 0$. By Theorem 4, a *necessary* condition is that $\exists(0, \beta_{n-3}, \tilde{\gamma}_{n-3}, \tilde{\gamma}_{n-2} = 1) \in G_{n-3}$ otherwise $|A\text{-consistent}| = 2^{t+3+c} > |\tilde{D}\text{-consistent}| = 2^{t+3}$ (invoke Theorem 2).

We now define a quantity $V_{n,t}$ which denotes the set of all $(x, y) \in \mathbb{Z}_2^n \times \mathbb{Z}_2^n$ such that the position of the least significant '1' of x is t ($V_{n,t}$ is a partition). We now define another quantity $l_{(n,t)}(k)$ for all $k \in [t+1, n-2]$. Let $l_{(n,t)}(k)$ denote a lower bound on the number of *adaptively chosen queries* to have $(0, \beta_i, \tilde{\gamma}_i, \tilde{\gamma}_{i+1} = 1) \in G_i$ for some $i \in [k, n-2]$ in the worst case of $(x, y) \in V_{n,t}$. In other words, a worst case lower bound $l_{(n,t)}(k)$ means that, for any algorithm $\mathcal{A}$ there exists a seed $(x, y) \in V_{n,t}$ such that the *adaptively chosen sequence* of $l_{(n,t)}(k) - 1$ queries ($l_{(n,t)}(k) > 0$) by $\mathcal{A}$ for the seed (x, y) produces oracle outputs $\tilde{\gamma}$'s with $\tilde{\gamma}_{i+1} = 0$, $\forall\, i \in [k, n-2]$). We will determine $l_{(n,t)}(n-3)$ which is a lower bound in this case. For easy reading, in the rest of the proof, we shall denote $l_{(n,t)}(k)$ by $l(k)$.

Let $p \in [t+1, n-3]$. Now, for each algorithm, we *always* identify a seed $(x, y) \in V_{n,t}$ such that the *adaptively chosen sequence* of $l(p) - 1$ queries produces $\tilde{\gamma}$'s with $\tilde{\gamma}_{i+1} = 0$, $\forall\, i \in [p, n-2]$. From Table 1, we construct (a, b), $(a', b') \in V_{n,t}$ for *each* (x, y) in the following fashion. The carry bit at the jth position of $(a + b)$ is c_j and similarly c'_j.

1. (Construction of a and b) $a_i = x_i$ and $b_i = y_i$, $\forall\, i \in [0, p]$. If $c_i = 0$ set $a_i = 0$, $b_i = 0$, $\forall\, i \in [p+1, n-1]$. If $c_i = 1$ set $a_i = 1$, $b_i = 1$, $\forall\, i \in [p+1, n-1]$.

2. (Construction of a' and b') $a'_i = x_i$ and $b'_i = y_i$, $\forall\, i \in [0, p]$. If $c'_i = 0$ set $a'_i = 0$, $b'_i = 1$, $\forall\, i \in [p+1, n-1]$. If $c'_i = 1$ set $a'_i = 1$, $b'_i = 0$, $\forall\, i \in [p+1, n-1]$.

It can be shown from the portion of Table 1 – cut off by the rows R(1) and R(3) and the columns Col(1), Col(2), Col(3) and Col(4) – that seeds (a, b) and (a', b') produce the same sequence of oracle outputs as (x, y) does on the selected sequence of $l(p) - 1$ queries.

Now we consider each possible $l(p)$th query $(0, \beta)$ and its output $\tilde{\gamma}$ for the seed (x, y). If $(\beta_{p+1}, \tilde{\gamma}_{p+1}) = (0, 1)$ then (a, b) produces $\tilde{\gamma}_{i+1} = 0$, $\forall\, i \in [p+1, n-2]$. Similarly, if $(\beta_{p+1}, \tilde{\gamma}_{p+1}) = (1, 1)$ then (a', b') produces $\tilde{\gamma}_{i+1} = 0$, $\forall\, i \in [p+1, n-2]$. If $(\beta_{p+1}, \tilde{\gamma}_{p+1}) = (0, 0)$ or $(1, 0)$ then both (a, b) and (a', b') produce $\tilde{\gamma}_{i+1} = 0$, $\forall\, i \in [p+1, n-2]$. Therefore, either (a, b) or (a', b') produces oracle outputs with $\tilde{\gamma}_{i+1} = 0$, $\forall\, i \in [p+1, n-2]$, for all the chosen $l(p)$ queries.

Thus, we establish that, for any algorithm there exists a seed $(x, y) \in V_{n,t}$ such that the *adaptively chosen* sequence of $l(p)$ queries produces oracle outputs $\tilde{\gamma}$'s with $\tilde{\gamma}_{i+1} = 0$, $\forall\, i \in [p+1, n-2]$. Therefore, a lower bound on the number of queries such that $\exists(0, \beta_i, \tilde{\gamma}_i, \tilde{\gamma}_{i+1} = 1) \in G_i$ for some $i \in [p+1, n-2]$ in the worst case of $(x, y) \in V_{n,t}$ is $l(p) + 1$. Therefore,

$$l(p+1) = l(p) + 1\,.$$

Following the recursion,

$$l(n-3) = n - t - 4 + l(t+1)\,. \tag{22}$$

The following lemma computes a value of $l(t+1)$. See Appendix A.2 for an elaborate proof.

Lemma 3 *Let $n-4 \geq t \geq 0$. For any algorithm $\mathcal{A}$ there exists a seed $(x,\, y) \in V_{n,\,t}$ such that the adaptively selected sequence of two queries by $\mathcal{A}$ for that particular seed produces oracle outputs $\tilde{\gamma}$'s with $\tilde{\gamma}_{i+1} = 0$, $\forall\, i \in [t+1,\, n-2]$.*

From Lemma 3, $l(t+1) = 3$. Therefore, from (22), $l(n-3) = n - t - 1$.
Case 2: $n = t+4$, a worst case lower bound is 3. The proof follows from Lemma 3.
Case 3: $n = t+3$, a worst case lower bound is 2. The reason is, from Table 1 it is clear that, with only one query $S_{n-2} > 1$ which makes the number of solutions for this case greater than 2^{t+3} which is impossible from Theorem 2.
Case 4: $n = t+2$, a worst case lower bound is 1 which is trivial. □

Theorem 6 *A lower bound on the number of queries $(\alpha,\, \beta)$ to solve*

$$(x+y) \oplus ((x \oplus \alpha) + (y \oplus \beta)) = \gamma$$

in the worst case of $(x,\, y) \in \mathbb{Z}_2^n \times \mathbb{Z}_2^n$ is (i) 3 if $n > 2$, (ii) 2 if $n = 2$, (iii) 0 if $n = 1$.

Proof. To prove *(i)*, we consider all possible *adaptively chosen* two queries by all algorithms and find that, in each case, an oracle can always select a seed such that the number of solutions obtained from the outputs exceeds 4 which is impossible by Theorem 3. A detailed analysis is given in Appendix A.4. The proofs for *(ii)* and *(iii)* are trivial. □

3.5 Optimal Algorithms

In this section, we concentrate on designing algorithms that solve (16) and (15) in the adaptive query model. In the formal framework, if the oracle is seeded with an unknown $(x,\, y)$ then there is always a *total useful set* $\tilde{D}$ that the unknown seed $(x,\, y)$ generates. For a particular *total useful set* $\tilde{D}$, there exists a set $\tilde{D}$-consistent containing all values of $(x,\, y)$. Our algorithm makes adaptive queries $(\alpha,\, \beta)$ to the oracle – which is already seeded with a fixed $(x,\, y)$ – and the oracle returns $\tilde{\gamma}$. The task of the algorithm is to compute $\tilde{D}$-consistent using oracle outputs $\tilde{\gamma}$, for all seeds $(x,\, y) \in \mathbb{Z}_2^n \times \mathbb{Z}_2^n$. The algorithm is *optimal* if the number of queries in the worst case (i.e., the upper bound) matches the lower bound derived in the relevant theorem (Theorem 5 or 6). Optimal algorithms to solve (16) and (15) are presented in Algorithm 2 and 3. Below we discuss the intuition behind the algorithms; while we omit many trivial details, the pseudocode covers all cases.

Discussion: Algorithm 2*(sketch).* The inputs to Algorithm 2 are an oracle O (which is set with the unknown seed and computes $\tilde{\gamma}$), n denoting the size of the unknowns and the precomputed Table 1 denoted by the variable T. The output of Algorithm 2 is a set of lists. This set contains the L_i's – the ith bit solution for all $i \in [0,\, n-2]$ – similar to the ones computed in Sect. 2.5. The objective of Algorithm 2 is to correctly compute the L_i's for the *useful set* $\tilde{D}$. The computed L_i's will be given to Algorithm 1 which computes all the individual solutions.

Algorithm 2 works in a way that at first G_i's are computed from the submitted queries such that the number of solutions for them is the same as that for all 2^n queries. Then the L_i's are computed from the G_i's (Step 28). First, we find t, the position of the least significant '1' of x, by making the first query $(0,\, \beta)$ with β composed of all 1's (Step 3, 10). Once the position of the least significant '1' of x is known, we can immediately compute G_i, $\forall i \in [0,\, t]$, by invoking Lemma 1 (Step 11 to 13). We see that $S_i = 2$, $\forall i \in [0,\, t]$. This result readily determines that $S_i = 1$, $\forall i \in [t+1,\, n-1]$ (see Theorem 2). To compute G_i, $\forall i \in [t+1,\, n-2]$ we will use a nice pattern observable in Table 1.

Algorithm 2 Optimal Algorithm to solve the equation $(x+y) \oplus (x + (y \oplus \beta)) = \gamma$

Require: Oracle O, n, Table T
Ensure: a set of lists

If $n \leq 0$ then exit with a comment "Invalid Input";
If $n = 1$ then return an empty set ϕ indicating that all solutions are possible and then exit;
$\beta = (1,\ 1,\ \cdots,\ 1,\ 1)_n$; /*The first query*/
$\tilde{\gamma} = O(\beta)$; /*Oracle output*/
if $\tilde{\gamma} = 0$
    For all $i \in [0,\ n-2]$
        $G_i = \{(0,\ 0,\ 0,\ 0),\ (0,\ 1,\ 0,\ 0)\}$;
    Go to Step 28;
$t = \mathsf{Least\text{-}Significant\text{-}one}(\tilde{\gamma})$;/*Computing the least significant '1' of the oracle output*/
$t = t - 1$; /*Computing the least significant '1' of x*/
For all $i \in [0,\ t-1]$ /*Computing G_i for all $i \in [0,\ t-1]$*/
    $G_i = \{(0,\ 0,\ 0,\ 0),\ (0,\ 1,\ 0,\ 0)\}$;
$G_t = \{(0,\ 0,\ 0,\ 0),\ (0,\ 1,\ 0,\ 1)\}$; /*Computing G_t*/
If $t = n - 2$ then Go to Step 28;
$\beta' = (1,\ 1,\ \cdots,\ 1,\ \beta'_{t+1} = 1,\ 0,\ 0,\ \cdots,\ 0)$; /*Query to make $\tilde{\gamma}'_{t+1} = 0$*/
$\tilde{\gamma}' = O(\beta')$; /*Oracle output*/
$G_{t+1} = \{(0,\ 1,\ 1,\ \tilde{\gamma}_{t+2}),\ (0,\ 1,\ 0,\ \tilde{\gamma}'_{t+2})\}$; /*Computing G_{t+1}*/
If $t = n - 3$ then Go to 28;
For all $i \in [2,\ n-t-2]$, in increasing order /*Computing G_i for all $i \in [t+2,\ n-2]$*/
    If $\tilde{\gamma}_{t+i} == \tilde{\gamma}'_{t+i} == 1$ then
        $\beta' = (1,\ 1,\ \cdots,\ 1,\ \beta'_{t+i-1} = 0,\ 0,\ \cdots,\ 0)$;/*Query to make $\tilde{\gamma}'_{t+i} = 0$*/
        $\tilde{\gamma}' = O(\beta')$ and Go to Step 27; /*Oracle output*/
    If $\tilde{\gamma}_{t+i} == \tilde{\gamma}'_{t+i} == 0$
        if $\tilde{\gamma}_{t+i-1} == 1$ then swap $((\beta,\ \tilde{\gamma}), (\beta',\ \tilde{\gamma}'))$;
        $\beta' = (1,\ 1,\ \cdots,\ 1,\ \beta'_{t+i-1} = 0,\ \beta'_{t+i-2},\ \cdots,\ \beta'_0)$;/*Query to make $\tilde{\gamma}'_{t+i} = 1$*/
        $\tilde{\gamma}' = O(\beta')$; /*Oracle output*/
    $G_{t+i} = \{(0,\ 1,\ \tilde{\gamma}_{t+i},\ \tilde{\gamma}_{t+i+1}),\ (0,\ 1,\ \tilde{\gamma}'_{t+i},\ \tilde{\gamma}'_{t+i+1})\}$;
$L_i = \{(x_i,\ y_i,\ c_i) \,|\, G_i \Rightarrow (x_i,\ y_i,\ c_i)\}$, $\forall i \in [0,\ n-2]$ (Using the table T);/*Computing L_i's*/
Return the set $\{L_i | i \in [0,\ n-2]\}$;

Consider Col (3) and Col (4) of the Table 1. Observe no two rows, contained in these two columns, are identical. Therefore, if for some queries G_i contains two elements of the form $(0, 1, 0, a)$ and $(0, 1, 1, b)$ then the corresponding $S_i = 1$. Therefore, the aim of Algorithm 2 is to submit queries such that G_i contains elements of the form $(0, 1, 0, a)$ and $(0, 1, 1, b)$, for all $i \in [t + 1, n - 2]$. Another interesting pattern of Table 1 is that each row, cut off by Col 2, Col 3 and Col 4, has at least one '1'. The algorithm uses this fact to get oracle output $\tilde{\gamma}$ with $\tilde{\gamma}_i = 1$ if required (Step 25). However, it is always easy to get an oracle output $\tilde{\gamma}$ with $\tilde{\gamma}_i = 0$ by setting the least i bits of β to zero (Step 15 and 21). Note that the algorithm submits the first two queries in Steps 3 and 15. In the loop (from steps 19 to 27) one of the two previous queries is modified to get '0' or '1' at the required locations of the output. Thus we get G_i, $\forall i \in [t + 1, n - 2]$, for which $S_i = 1$. Therefore, invoking Theorem 2, we get the number solutions for the submitted queries is 2^{t+3} (omitting many trivial cases). This proves the correctness of Algorithm 2. It can be easily verified from Theorem 5 that the number of queries used by Algorithm 2 is worst case optimal.

Discussion: Algorithm 3 *(sketch).* The basic idea of Algorithm 3 is the same as that of Algorithm 2, i.e., we compute the L_i's for the *total useful set* $\tilde{D}$. But here we use a different trick to optimize the number of queries. Note that, for all $i \in [0, n-2]$, $S_i = 1$ for (15) (see Theorem 3).

We first submit two queries as shown in Step 3 and 5. For these two queries $G_i = \{(1, 0, \tilde{\gamma}_i, \tilde{\gamma}_{i+1}), (0, 1, \tilde{\gamma}'_i, \tilde{\gamma}'_{i+1})\}$ if i even. Note that, in this case, if $\tilde{\gamma}_i = \tilde{\gamma}'_i$ then $S_i = 1$ otherwise $S_i = 2$ (from Table 1). $G_i = \{(1, 0, \tilde{\gamma}_i, \tilde{\gamma}_{i+1}), (1, 0, \tilde{\gamma}'_i, \tilde{\gamma}'_{i+1})\}$ if i odd. Note that, in this case, if $\tilde{\gamma}_i \neq \tilde{\gamma}'_i$ then $S_i = 1$ otherwise $S_i = 2$. Observe that $S_i = 2 \Leftrightarrow |L_i| = 4$ and $S_i = 1 \Leftrightarrow |L_i| = 2$. Now if $|L_i| = 2$, $\forall i \in [0, n - 2]$ then we are done (Step 11). A combinatorial pattern in the precomputed Table 1 shows that, if $|L_i| = 4$ then $|L_{i-1}| = 2$ (omitting the proof). In Step 13 and 16, we change the second query (c, d) to obtain the third query so that the new query can remove extra elements from the L_i's with $|L_i| = 4$. The rule is if $|L_i| = 4$ then change the $(i-1)$th bits of (c, d) "suitably" (details in the pseudocode). In Steps 13 and 16, the relation operator "$\Rightarrow$" is used in the same sense as in Sect. 2.3. Now $L'_i s$ are computed from the first and the third queries (Step 21). If $|L_i| = 4$ then we assign $L_i = L'_i$ (Step 23). Now we get $|L_i| = 2$, $\forall i \in [0, n - 2]$. Algorithm 3 is *optimal* because it uses maximum 3 queries.

Time and Memory. For each of Algorithm 2 and Algorithm 3, the memory and the time are $\Theta(n)$ and $\mathcal{O}(n)$ respectively (the oracle takes $\mathcal{O}(1)$-time to compute $\tilde{\gamma}$).

4 Cryptographic Applications: Attacking the Helix Cipher

Weakness of modular addition under DC. The fact that an arbitrary system of DEA is in the complexity class P, shows a major differential weakness of modular addition which is one of the most used block cipher components. The weakness is more alarming because, with a maximum of only 3 adaptively chosen queries the search space of the secret can be reduced from 2^{2n} to only 4 for all $n \geq 1$ (see Theorem 3 and Algorithm 3). It is, however, not known at this moment, how much influence this weakness has on many modern symmetric ciphers that mix other nonlinear operations with addition. Still, a recommendation for a deeper analysis of ciphers mixing XOR and modular addition in the light of our investigation is well justified.

Cryptanalysis of Helix. Helix, proposed by Ferguson *et al.* [FW+03], is a stream cipher with a combined MAC functionality. The primitive uses combination of addition and XOR to generate pseudorandom bits. Recently a differential attack was found against Helix by Muller [Mul04]. He solved the equation $(x + y) \oplus (x + (y \oplus \beta)) = \gamma$ many times to recover secret information (x, y) using β and the corresponding γ. Every time β corresponds to a chosen plaintext. His algorithm uses $3(n - 1)$ queries every time. Therefore, the most natural challenge, from an algorithmic point

Algorithm 3 Optimal algorithm to solve the equation $(x + y) \oplus ((x \oplus \alpha) + (y \oplus \beta)) = \gamma$

Require: Oracle O, n, Table T

Ensure: a set of lists

1: If $n \leq 0$ then exit with a comment "Invalid Input";
2: If $n = 1$ then return an empty set ϕ indicating that all solutions are possible and then exit;
3: $(a, b) = ((11 \cdots 11)_n, (00 \cdots 00)_n)$;/*First Query*/
4: $\tilde{\gamma} = O(a, b)$; /*Oracle Output*/
5: $(c, d) = ((\cdots 101010)_n, (\cdots 010101)_n)$;/*Second Query*/
6: $\tilde{\gamma}' = O(c, d)$; /*Oracle Output*/
7: For all $i \in [0, n-2]$ /*Computing G_i's*/
8: $G_i = \{(a_i, b_i, \tilde{\gamma}_i, \tilde{\gamma}_{i+1}), (c_i, d_i, \tilde{\gamma}'_i, \tilde{\gamma}'_{i+1})\}$;
9: For all $i \in [0, n-2]$ /*Computing L_i's*/
10: $L_i = \{(x_i, y_i, c_i) \,|\, G_i \Rightarrow (x_i, y_i, c_i)\}$, $\forall i \in [0, n-2]$ (Using the table T);
11: If $|L_i| = 2$, $\forall i \in [0, n-2]$ then Go to step 25;
12: For all $i \in [0, n-2]$ and i even /*Checking even L_i's*/
13: If $|L_i| = 4$ then collect $(x_{i-1}, y_{i-1}, 0) \in L_{i-1}$ and $(1, 0, \tilde{\gamma}_{i-1}, \tilde{\gamma}_i) \in G_{i-1}$.
Select $(\alpha_{i-1}, \beta_{i-1})$ from T such that
$\{(\alpha_{i-1}, \beta_{i-1}, 0, \tilde{\gamma}_i), (\alpha_{i-1}, \beta_{i-1}, 1, \tilde{\gamma}_i)\} \Rightarrow (x_{i-1}, y_{i-1}, 0)$;
14: $(c_{i-1}, d_{i-1}) = (\alpha_{i-1}, \beta_{i-1})$;
15: For all $i \in [0, n-2]$ and i odd /*Checking odd L_i's*/
16: If $|L_i| = 4$ then collect $(x_{i-1}, y_{i-1}, 0) \in L_{i-1}$ and $(1, 0, \tilde{\gamma}_{i-1}, \tilde{\gamma}_i) \in G_{i-1}$.
Select $(\alpha_{i-1}, \beta_{i-1})$ from T such that
$\{(\alpha_{i-1}, \beta_{i-1}, 0, 1 \oplus \tilde{\gamma}_i), (\alpha_{i-1}, \beta_{i-1}, 1, 1 \oplus \tilde{\gamma}_i)\} \Rightarrow (x_{i-1}, y_{i-1}, 0)$;
17: $(c_{i-1}, d_{i-1}) = (\alpha_{i-1}, \beta_{i-1})$;
18: $\tilde{\gamma}' = O(c, d)$;/*Oracle Output*/
19: For all $i \in [0, n-2]$ /*Computing G_i's*/
20: $G_i = \{(a_i, b_i, \tilde{\gamma}_i, \tilde{\gamma}_{i+1}), (c_i, d_i, \tilde{\gamma}'_i, \tilde{\gamma}'_{i+1})\}$;
21: For all $i \in [0, n-2]$ /*Computing L'_i's*/
22: $L'_i = \{(x_i, y_i, c_i) \,|\, G_i \Rightarrow (x_i, y_i, c_i)\}$, $\forall i \in [0, n-2]$ (Using the table T);
23: For each $i \in [0, n-2]$
24: If $|L_i| = 4$, then assign $L_i = L'_i$;
25: Return the set $\{L_i | i \in [0, n-2]\}$;

of view, is to reduce the number of queries and if possible to attain an optimality. For the Helix output word $n = 32$ bits, 93 queries were needed whereas Algorithm 2 takes *at most* 31 queries if the position of the least significant '1' of x (denoted by t) is zero. Note that, if $t > 0$ then the number of queries is less. However, the most important fact is that the number of queries cannot be further reduced in the worst case as our algorithm is worst case optimal. This fact can be straightaway used to reduce the data complexity of that particular attack on Helix cipher by, *at least*, a factor of 3. However, in the best case, there exists seed (x, y), $\forall\, t \in [0, n-3]$, for which (16) can be solved by Algorithm 2 with only 2 queries and the improvement in such a case is a factor of 46.5.

Computing Differential Properties of Addition. It is easy to show that our algorithm to solve a randomly generated set DEA is robust enough to compute all differential properties of addition (e.g., maximal differential, impossible differential, etc.) In such a case we need to consider a single differential equation of addition instead of many and then compute the solutions to it. For example, if the single equation is NOT satisfiable then this is an impossible differential. However, the differential properties of addition has been independently studied in depth by Lipmaa and Moriai using a different approach [LM02].

5 Conclusion and Further Research

The results of the paper have impacts on both theory and practice. We have shown the importance of solving DEA from both a theoretical and a practical point of view. The paper seals any further search to improve lower bounds on the number of queries for solving DEA with adaptive queries. Although the total number of queries grows exponentially, an optimal lower bound is linear for one of them and constant for the other. Moreover, our algorithm reduces the number of queries of the previous best known algorithm and our results improve the data complexity of an attack on Helix cipher. Our work leaves room for further research, particularly, in the direction of solving DEA with nonadaptive queries. Last but not the least, our solution techniques motivate further research to solve more complex equations that mix *modular addition*, *exclusive-or*, *modular multiplication* and *T-functions*.

6 Acknowledgments

We are grateful to Mridul Nandi of the Indian Statistical Institute for encouraging us to do this work and for his critical remarks on many technical results. We also thank Taizo Shirai of Sony Incorporation, Japan, Sankardas Roy of George Mason University, USA and Jongsung Kim of Katholieke Universiteit Leuven, Belgium for numerous useful discussions. Thanks are also due to the anonymous reviewers of ACISP 2005 for their constructive comments.

References

[AH+] A. Aho, J. Hopcroft, J. Ullman, "The Design and Analysis of Computer Algorithms," Addison-Wesley, 1974.

[AL95] I. A. Ajwa, Z. Liu, P. S. Wang, "Gröbner Bases Algorithm," *ICM Technical Report*, February 1995, Available Online at `http://icm.mcs.kent.edu/reports/1995/gb.pdf`. (Cited on page 5.)

[Ber92] T. A. Berson, "Differential Cryptanalysis Mod 2^{32} with Applications to MD5," *Eurocrypt 1992* (R. A. Rueppel, ed.), vol. 658 of *LNCS*, pp. 71-80, Springer-Verlag, 1993. (Cited on pages 4 and 5.)

[BC+98] C. Burwick, D. Coppersmith, E. D'Avignon, Y. Gennaro, S. Halevi, C. Jutla, S. M. Matyas Jr., L. O'Connor, M. Peyravian, D. Safford and N. Zunic, "MARS – A Candidate Cipher for AES," Available Online at `http://www.research.ibm.com/security/mars.html`, June 1998. (Cited on page 3.)

[BS91] E. Biham, A. Shamir, "Differential Cryptanalysis of DES-like Cryptosystems," *Crypto '90* (A. Menezes, S. A. Vanstone, eds.), vol. 537 of *LNCS*, pp. 2-21, Springer-Verlag, 1991. (Cited on page 3.)

[CL+90] T. H. Cormen, C. E. Leiserson, R. L. Rivest, *"Introduction to Algorithms,"* MIT Press. (Cited on page 3.)

[Fau99] J. Faugère, "A new effecient algorithm for computing Gröbner bases (F_4)," *Journal of Pure and Applied Algebra*, vol. 139, pp. 61-88, 1999, Available Online at `http://www.elsevier.com/locate/jpaa`. (Cited on pages 4 and 5.)

[FB] R. Floyd, R. Beigel, "The Language of Machines,"W. H. Freeman, 1994. (Cited on page 5.)

[FW+03] N. Ferguson, D. Whiting, B. Schneier, J. Kelsey, S. Lucks, T. Kohno, "Helix: Fast Encryption and Authentication in a Single Cryptographic Primitive," *Fast Software Encryption 2003* (T. Johansson, ed.), vol. 2887 of *LNCS*, pp. 330-346, Springer-Verlag, 2003. (Cited on page 5.)

[HM+04] J. E. Hopcroft, R. Motwani, J. D. Ullman, *''Introduction to Automata Theory, Languages and Computation,"* Second Edition, Pearson Education, 2004. (Cited on pages 3, 4 and 15.)

[Knu81] D. E. Knuth, *"The Art of Computer Programming," vol. 2, Seminumerical Algorithms,* Addison-Wesley Publishing Company, 1981. (Cited on page 5.)

[KS03] A. Klimov, A. Shamir, "Cryptographic Applications of T-Functions," *Selected Areas in Cryptography 2003* (M. Matsui, R. J. Zuccherato, eds.), vol. 3006 of *LNCS*, pp. 248-261, Springer-Verlag, 2004. (Not cited.)

[KS04] A. Klimov, A. Shamir, "New Cryptographic Primitives Based on Multiword T-Functions," *Fast Software Encryption 2004* (B. Roy, W. Meier, eds.), vol. 3017 of *LNCS*, pp. 1-15, Springer-Verlag, 2004. (Cited on page 3.)

[KS99] A. Kipnis, A. Shamir, "Cryptanalysis of the HFE Public Key Cryptosystems by Relinearization," *Crypto 1999* (M. Wiener, ed.), vol. 1666 of *LNCS*, pp. 19-30, Springer-Verlag, 1999. (Cited on page 3.)

[LM+91] X. Lai, J. L. Massey, S. Murphy, "Markov Ciphers and Differential Cryptanalysis," *Eurocrypt '91* (W. Davis, ed.), vol. 547 of *LNCS*, pp. 17-38, Springer-Verlag, 1991. (Cited on page 5.)

[LM02] H. Lipmaa, S. Moriai, "Efficient Algorithms for Computing Differential Properties of Addition," *FSE 2001* (M. Matsui, ed.), vol. 2355 of *LNCS*, pp. 336-350, Springer-Verlag, 2002. (Cited on page 3.)

[LW+04] L. Lipmaa, J. Wallén, P. Dumas, "On the Additive Differential Probability of Exclusive-Or," *Fast Software Encryption 2004* (B. Roy, W. Meier, eds.), vol. 3017 of *LNCS*, pp. 317-331, Springer-Verlag, 2004. (Cited on pages 3, 4 and 17.)

[Mul04] F. Muller, "Differential Attacks against the Helix Stream Cipher," *Fast Software Encryption 2004* (B. Roy, W. Meier, eds.), vol. 3017 of *LNCS*, pp. 94-108, Springer-Verlag, 2004. (Cited on page 3.)

[NK91] K. Nyberg, L. Knudsen, "Provable Security Against a Differential Attack," *Journal of Cryptology*, 8(1):27-37, 1991. (Cited on pages 3, 4, 9 and 15.)

[RR+99] R. L. Rivest, M. Robshaw, R. Sidney, Y. L. Yin, "The RC6 Block Cipher," Available Online at `http://theory.lcs.mit.edu/ rivest/rc6.ps`, June 1998. (Cited on page 3.)

[SK+99] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C.Hall, N. Ferguson, "The Twofish Encryption Algorithm: A 128-Bit Block Cipher," John Wiley & Sons, April 1999, ISBN: 0471353817. (Cited on page 3.)

[SM90] O. Staffelbach, W. Meier, "Cryptographic Significance of the Carry for Ciphers Based on Integer Addition," *Crypto '90* (A. Menezes, S. A. Vanstone, eds.), vol. 537 of *LNCS*, pp. 601-614, Springer-Verlag, 1991. (Cited on page 3.)

[YB+04] H. Yoshida, A. Biryukov, Christophe De Cannière, J. Lano, B. Preneel, "Non-randomness of the Full 4 and 5-Pass HAVAL," *SCN 2004* (Carlo Blundo and Stelvio Cimato, eds.), vol. 3352 of *LNCS*, pp. 324-336, Springer-Verlag, 2004. (Cited on page 3.)

[Wal03] J. Wallén, "Linear Approximations of Addition Modulo 2^n," *Fast Software Encryption 2003* (T. Johansson, ed.), vol. 2887 of *LNCS*, pp. 261-273, Springer-Verlag, 2003.

A Appendix

A.1 Proofs of Lemma 1 and 2

Claim 1 *For all* $(0, \beta, \tilde{\gamma}) \in \tilde{D}$, $\tilde{\gamma}_i = 0 \; \forall \, i \in [0, t]$.

Proof. If the position of the least significant '1' of x is t then $c_i = \tilde{c}_i = 0 \; \forall \, i \in [0, t]$ and $\forall \, \beta \in \mathbb{Z}_2^n$ (see (4)). Recall $\tilde{\gamma}_i = c_i \oplus \tilde{c}_i$. This proves the lemma. □

Claim 2 *For each* $i \in [t+1, n-1]$, *there exists* $(0, \beta, \tilde{\gamma}) \in \tilde{D}$ *with* $\tilde{\gamma}_i = 1$.

Proof. We prove the lemma by induction on i. The statement is true if $i = t+1$. Suppose, the statement is true if $i = k$ for some $k \in [t+1, n-2]$, that is, there exists $(0, a, b) \in \tilde{D}$ with $b_k = 1$ (induction hypothesis). We construct three n-bit integers from a,

1. $a' = (a_{n-1}, a_{n-2}, \cdots, a_{k+1}, 0, a_{k-1}, \cdots, a_0)$
2. $a'' = (a_{n-1}, a_{n-2}, \cdots, a_{k+1}, 1, a_{k-1}, \cdots, a_0)$
3. $a''' = (a_{n-1}, a_{n-2}, \cdots, a_{k+1}, 1, 0, 0, \cdots, 0)$.

Now we select three elements $(0, a', b')$, $(0, a'', b'')$, $(0, a''', b''') \in \tilde{D}$ (such elements exist since, for all $p \in \mathbb{Z}_2^n$, there exists $(0, p, q) \in \tilde{D}$ for some $q \in \mathbb{Z}_2^n$). Note that $b'_k = b''_k = b_k = 1$ and $b'''_k = 0$. From Table 1, at least one of b'_{k+1}, b''_{k+1} and b'''_{k+1} is 1. This proves the lemma. □

A.2 Proof of Lemma 3

Claim 3 *Let* $n - 4 \geq t \geq 0$. *For any algorithm* $\mathcal{A}$ *there exists a seed* $(x, y) \in V_{n,\,t}$ *such that the adaptively selected sequence of two queries by* $\mathcal{A}$ *for that particular seed produces oracle outputs* $\tilde{\gamma}$*'s with* $\tilde{\gamma}_{i+1} = 0$, $\forall \, i \in [t+1, n-2]$.

Proof. Let the first two queries and the corresponding oracle outputs be $(0, \beta)$, $(0, \beta')$, $\tilde{\gamma}$ and $\tilde{\gamma}'$. Depending *only* on the tth bit of β and β', the oracle returns outputs (i.e., $\tilde{\gamma}$ and $\tilde{\gamma}'$) according to the following rules.

1. If $\beta_t = 0$ then the oracle returns $\tilde{\gamma} = (0, 0, \cdots, 0)_n$. (Cited on page 3.)
2. If $\beta_t = 1$ then $\tilde{\gamma}_{t+1} = 1$ and all other bits of $\tilde{\gamma}$ are zero. (Cited on page 3.)

3. If $\beta'_t = 0$ then the oracle returns $\tilde{\gamma}' = (0,\, 0,\, \cdots,\, 0)_n$.

4. If $\beta'_t = 1$ then $\tilde{\gamma}'_{t+1} = 1$ and all other bits of $\tilde{\gamma}'$ are zero.

Under any of the above input-output combinations there exists a seed and $\tilde{\gamma}_{i+1} = 0\ \forall\, i \in [t{+}1,\, n{-}2]$ for all outputs $\tilde{\gamma}$. This proves the lemma. □

A.3 Construction of M from the L_i's

Let G_i be known $\forall\, i \in [0,\, n-2]$ $(n > 1)$ for a nonempty *useful set* $\tilde{A}$. Let $L_i = \{(x_i,\, y_i,\, c_i)\,|\,G_i \Rightarrow (x_i,\, y_i,\, c_i)\}\ \forall i \in [0,\, n-2]$. Let a set M be constructed from the L_i's in the following way,

$$\begin{aligned} M \;=\; & \{((x_{n-1},\, x_{n-2},\, \cdots,\, x_0),\, (y_{n-1},\, y_{n-2},\, \cdots,\, y_0)) \\ & |(x_{n-1},\, y_{n-1}) \in \mathbb{Z}_2^2,\, (x_i,\, y_i,\, c_i) \in L_i,\, i \in [0,\, n-2],\, c_0 = 0\,, \\ & c_{i+1} = x_i y_i \oplus x_i c_i \oplus y_i c_i\}\,. \end{aligned}$$

We present an algorithm to construct M from the L_i's with memory $\mathcal{O}(n \cdot S)$ and time $\mathcal{O}(S)$ where $S = |\tilde{A}\text{-consistent}|$.

First we set

$$M_1 = \{((c_1),\, (x_0),\, (y_0))|(x_0,\, y_0,\, 0) \in L_0,\, c_1 = x_0 y_0\}\,.$$

Now we construct a set $M_k\ \forall\, k \in [2,\, n-1]$ using the following recursion.

$$\begin{aligned} M_k \;=\; & \{((c_k),\, (x_{k-1},\, \cdots,\, x_0),\, (y_{k-1},\, \cdots,\, y_0))|(x_{k-1},\, y_{k-1},\, c_{k-1}) \in L_{k-1}\,, \\ & ((c_{k-1}),\, (x_{k-2},\, \cdots,\, x_0),\, (y_{k-2},\, \cdots,\, y_0)) \in M_{k-1},\, c_k = x_{k-1}y_{k-1} \oplus x_{k-1}c_{k-1} \oplus y_{k-1}c_{k-1}\}\,. \end{aligned}$$

Now, we construct

$$\begin{aligned} M_n \;=\; & \{((x_{n-1},\, \cdots,\, x_0),\, (y_{n-1},\, \cdots,\, y_0))|(x_{n-1},\, y_{n-1}) \in \mathbb{Z}_2^2\,, \\ & ((c_{n-1}),\, (x_{n-2},\, \cdots,\, x_0),\, (y_{n-2},\, \cdots,\, y_0)) \in M_{n-1}\}\,. \end{aligned}$$

Using Proposition 2 and Theorem 1 it is easy to show that $M = M_n$. Note that the size of each L_i is $\mathcal{O}(1)$ since the size of the Table 1 is $\mathcal{O}(1)$. Also note that $|M_n| = S$ and therefore the asymptotic memory requirement to construct M_n recursively following the above algorithm is $\mathcal{O}(n \cdot S)$ since $k = \mathcal{O}(n)$ and M_{k+1} can be constructed from M_k only. It is trivial to show that the time to construct M_n (i.e., M) from the L_i's is $\mathcal{O}(S)$. Thus, the set M can be constructed from the L_i's with memory $\mathcal{O}(n \cdot S)$ and time $\mathcal{O}(S)$.

A.4 Proof of Lower Bound for (3)

Claim 4 *A lower bound on the number of queries* $(\alpha,\, \beta)$ *to solve*

$$(x + y) \oplus ((x \oplus \alpha) + (y \oplus \beta)) = \gamma$$

in the worst case of $(x,\, y) \in \mathbb{Z}_2^n \times \mathbb{Z}_2^n$ *is*
(i) 3 *if* $n > 2$,
(ii) 2 *if* $n = 2$,
(iii) 0 *if* $n = 1$.

Proof. *(i)* $n > 2$. Let the first two queries and the corresponding oracle outputs be $(\alpha,\, \beta)$, $(\alpha',\, \beta')$, $\tilde{\gamma}$ and $\tilde{\gamma}'$. Depending on the two least significant bits of α, β, α' and β', the oracle returns outputs (i.e., $\tilde{\gamma}$ and $\tilde{\gamma}'$) according to the following rules.

1. If $(\alpha_0, \beta_0) = (0, 0)$ then $\tilde{\gamma} = (0, 0, \cdots, 0)_n$.
2. If $(\alpha_0, \beta_0) \neq (0, 0)$ and $(\alpha_1, \beta_1) = (1, 1)$ then $\tilde{\gamma} = (1, 1, \cdots, 1, 0)_n$.
3. If $(\alpha_0, \beta_0) \neq (0, 0)$ and $(\alpha_1, \beta_1) \neq (1, 1)$ then $\tilde{\gamma} = (0, 0, \cdots, 0)_n$.
4. If $(\alpha_0, \beta_0) = (\alpha'_0, \beta'_0)$ then $\tilde{\gamma} = \tilde{\gamma}'$.
5. If $(\alpha_0, \beta_0) \neq (\alpha'_0, \beta'_0) = (0, 0)$ then $\tilde{\gamma}' = (0, 0, \cdots, 0)_n$.
6. If $(\alpha_0, \beta_0) \neq (\alpha'_0, \beta'_0) \neq (0, 0)$ and $(\alpha'_1, \beta'_1) = (0, 0)$ then $\tilde{\gamma}' = (0, 0, \cdots, 0)_n$.
7. If $(\alpha_0, \beta_0) \neq (\alpha'_0, \beta'_0) \neq (0, 0)$ and $(\alpha'_1, \beta'_1) = (1, 1)$ then $\tilde{\gamma}' = (1, 1, \cdots, 1, 0)_n$.
8. If $(\alpha_0, \beta_0) \neq (\alpha'_0, \beta'_0) \neq (0, 0)$, $(\alpha'_1, \beta'_1) \in \{(0, 1), (1, 0)\}$ and $(\alpha_1, \beta_1) = (\alpha'_1, \beta'_1)$ then $\tilde{\gamma}' = (0, 0, \cdots, 0)_n$.
9. If $(\alpha_0, \beta_0) \neq (\alpha'_0, \beta'_0) \neq (0, 0)$, $(\alpha'_1, \beta'_1) \in \{(0, 1), (1, 0)\}$ and $(\alpha_1, \beta_1) \neq (\alpha'_1, \beta'_1)$ then $\tilde{\gamma}'_0 = 0$ and $\tilde{\gamma}'_i = 1 \oplus \tilde{\gamma}_i$ for all $i \in [1, n-1]$.

From the oracle outputs produced according to the above rules on the first two queries, one can show, using Table 1, that one of the following cases occurs.

1. $S_0 \geq 2$ and $S_i \geq 1 \; \forall \, i \in [0, n-2]$.
2. $S_1 \geq 2$ and $S_i \geq 1 \; \forall \, i \in [0, n-2]$.
3. $S_0 \geq 2$, $S_1 \geq 2$ and $S_i \geq 1 \; \forall \, i \in [0, n-2]$.

Clearly, for any of the above cases, the number of valid solutions S, derived from the results of the queries, is *at least* 8 which is not the case with this equation (see Theorem 3). Therefore, a lower bound on the number of queries in the worst case is 3.
(ii) $n = 2$. Using Table 1, a proof is similar to the proof for *(i)*.
(iii) $n = 1$. A proof is trivial. □