EasyBC: A Cryptography-Specific Language for Security Analysis of Block Ciphers against Differential Cryptanalysis

PU SUN, ShanghaiTech University, China
FU SONG*, Institute of Software at Chinese Academy of Sciences, China and University of Chinese Academy of Sciences, China
YUQI CHEN, ShanghaiTech University, China
TAOLUE CHEN, Birkbeck, University of London, UK

Differential cryptanalysis is a powerful algorithmic-level attack, playing a central role in evaluating the security of symmetric cryptographic primitives. In general, the resistance against differential cryptanalysis can be characterized by the maximum expected differential characteristic probability. In this paper, we present generic and extensible approaches based on mixed integer linear programming (MILP) to bound such probability. We design a high-level cryptography-specific language EasyBC tailored for block ciphers and provide various rigorous procedures as differential denotational semantics, to automate the generation of MILP from block ciphers written in EasyBC. We implement an open-sourced tool that provides support for fully automated resistance evaluation of block ciphers against differential cryptanalysis. The tool is extensively evaluated on 23 real-life cryptographic primitives including all the 10 finalists of the NIST lightweight cryptography standardization process. The experiments confirm the expressivity of EasyBC and show that the tool can effectively prove the resistance against differential cryptanalysis for all block ciphers under consideration. EasyBC makes resistance evaluation against differential cryptanalysis easily accessible to cryptographers.

CCS Concepts: • **Theory of computation**; • **Security and privacy**;

Additional Key Words and Phrases: Cryptography-Specific Language, Block Ciphers, Differential Cryptanalysis

ACM Reference Format:
Pu Sun, Fu Song, Yuqi Chen, and Taolue Chen. 2024. EasyBC: A Cryptography-Specific Language for Security Analysis of Block Ciphers against Differential Cryptanalysis. *Proc. ACM Program. Lang.* 8, POPL, Article 29 (January 2024), 34 pages. https://doi.org/10.1145/3632871

1 INTRODUCTION

A block cipher is a symmetric cryptographic technique that uses the same key to encrypt and decrypt data in fixed-size blocks. Guided by the design principles of block ciphers [Shannon 1949], a vast number of block ciphers have been proposed, varying in, e.g., network structures and block sizes. Many of them have been standardized and are widely used in daily life to provide confidentiality, integrity, and authentication. Differential cryptanalysis, proposed by [Biham and Shamir 1990], is a powerful algorithmic-level attack against block ciphers by analyzing the effect of particular differences in input pairs on the differences of pairs of intermediate states under

*Corresponding author

Authors' addresses: Pu Sun, ShanghaiTech University, Shanghai, China, sunpu@shanghaitech.edu.cn; Fu Song, Institute of Software at Chinese Academy of Sciences, Beijing, China and University of Chinese Academy of Sciences, Beijing, China, songfu@ios.ac.cn; Yuqi Chen, ShanghaiTech University, Shanghai, China, chenyq@shanghaitech.edu.cn; Taolue Chen, Birkbeck, University of London, London, UK, t.chen@bbk.ac.uk.

ACM 2475-1421/2024/1-ART29
https://doi.org/10.1145/3632871

the same key. It has proved to be a very effective attack, which has broken block ciphers such as DES [Biham and Shamir 1990], FEAL [Aoki et al. 1997], WARP [Teh and Biryukov 2022], and K-Cipher [Mahzoun et al. 2022]. As a result, a provable guarantee of the resistance of block ciphers against differential cryptanalysis has become a standard criterion for new block ciphers and indeed a basic requirement for them to be standardized [Katz and Lindell 2014]. In light of the diversity and wide deployment of block ciphers, a generic, and ideally automated, approach which can be used to evaluate their resistance against differential cryptanalysis, becomes indispensable.

In general, the resistance of block ciphers against differential cryptanalysis is commonly characterized by the maximum expected differential characteristic probability (MaxEDCP for short, the definition of which is fairly standard but technical, and will be given in Section 2.1). A block cipher is considered to be resistant against differential cryptanalysis if the MaxEDCP is no greater than $2^{-\mathscr{b}}$, where $\mathscr{b}$ is the block size of the block cipher [Heys 2002; Lai et al. 1991]. In light of this, the central task of security analysis for block ciphers against differential cryptanalysis is reduced to computing such probability. To this end, [Matsui 1994] proposed a branch-and-bound searching algorithm that traverses differential characteristics in a depth-first manner and computes their probabilities during traversal. However, it becomes inefficient with the increasing of candidate differential characteristics. Various heuristics are then proposed to improve the efficiency [Aoki et al. 1997; Bao et al. 2014; Biryukov and Nikolić 2010; Ji et al. 2021], but they harness cipher-specific optimizations and thus require sophisticated programming skills.

Several alternative methods are introduced, which reduce to mixed integer linear program (MILP [Mouha et al. 2011]), Boolean satisfiability problem (SAT [Mouha and Preneel 2013]), or satisfiability modulo theory (SMT [Aumasson et al. 2014]). These methods allow cryptanalysts to specify the problem for each cipher using the input language of MILP/SAT/SMT solvers so that respective solvers can be harnessed. Plenty of modeling methods for cryptography-specific operations such as substitution-box (S-box), Exclusive-OR (XOR), and linear transformations, as well as heuristics, are proposed to improve efficiency and accuracy. However, insofar cryptanalysts have to manually model each cipher in the tool they choose to use, or at best write a model generation script for each cipher, which is usually intricate, error-prone, and laborious, as it requires the cryptanalysts to be familiar with the specific tool and a wide range of modeling methods. There appears to be a lack of language support, unified computational approaches, and fully automated tools for evaluating the resistance of block ciphers against differential cryptanalysis.

Contributions. In this work, our primary aim is to develop a generic and automated approach for evaluating the resistance of block ciphers against differential cryptanalysis. To achieve this goal, we begin by designing a novel high-level statically-typed, C-like cryptography-specific language, EasyBC (**Easy B**lock **C**ipher), tailored for block ciphers. Besides standard types and operations, EasyBC provides cryptography-specific types (e.g., S-boxes, P-boxes) and operations (e.g., substitution via S-box, linear transformation via P-box, or matrix-vector product), to facilitate the implementation of block ciphers. (Note that an S-box takes m input bits and transforms them into n output bits and P-box, short for Permutation-box, is an array specifying a permutation of inputs.) The language is fully specified by a formal grammar together with typing rules and operational semantics, enabling further automatic analysis.

Concretely speaking, the analysis of block cipher resistance against differential cryptanalysis primarily involves computing the MaxEDCP. However, calculating MaxEDCP precisely is practically infeasible. As a result, we employ a strategy to calculate a tight upper bound that is sufficient to demonstrate resistance. Specifically, we adopt the typical MILP-based approach where linear constraints are used to characterize the dependency (i.e., feasibility) between input and output differences of each operation, and the optimization objective is to minimize an upper bound. In particular, we give a rigorous procedure, formalized as a differential denotational semantics, to automate the generation of MILP from EasyBC programs, which not only unifies and optimizes the existing but also discovers new generation processes. Our approach is of generic nature, thanks to the expressivity of the EasyBC langauge, namely, a multitude of block ciphers can be handled in a unified way.

Technically, the generation of MILP can be done either at the word or bit level. The former is less involved and generates fewer constraints, but is limited to certain block ciphers; the latter approach is more fine-grained and has wider applicability. In both cases, the general strategy is to determine the lower bound of the minimum number of (differentially) active S-boxes, i.e., S-boxes whose input differences are nonzero under two executions [Biryukov and Nikolić 2010; Heys 2002], from which the upper bound of the MaxEDCP can be deduced according to [Heys 2002; Sun et al. 2014a]. While this strategy is efficient, it is important to note that the obtained upper bound may not always be sufficiently tight and may not be applicable for certain ciphers. To address this limitation, we introduce an extended bit-wise approach that directly bounds the MaxEDCP by encoding probabilities using additional Boolean variables. We have successfully implemented our approach as the first fully automated tool for evaluating the resistance of block ciphers against differential cryptanalysis. This tool eliminates the need for cryptanalysts to possess knowledge of MILP generation for cryptographic operations. Instead, they can simply write a program in EasyBC for a block cipher. Moreover, the generation of MILP from EasyBC program is modular, i.e., each cryptographic operation in EasyBC is associated with its own MILP generation rule and new generation rules could be easily added by implementing designated APIs. As a result, our approach exhibits excellent extensibility for new block ciphers, enabling a wider range of applicability.

To evaluate the tool, we implement 23 realistic cryptographic primitives with EasyBC, including all the 10 finalists of the NIST lightweight cryptography standardization process [NIST 2023] and other commonly used block ciphers, covering both substitution-permutation network (SPN) based ciphers (e.g., AES, PRESENT, and GIFT) and balanced Feistel networks (BFN) based ciphers (e.g., DES, LBLOCK, and TWINE). It turns out that EasyBC can express these block ciphers in a considerably more succinct way, demonstrating our language's expressiveness. Moreover, it turns out that our tool is able to effectively handle all realistic block ciphers under consideration, showcasing its capability in proving the resistance of block ciphers against differential cryptanalysis.

In addition, we compare various alternative MILP generation methods for cryptographic operations. Interestingly, we observe that certain methods may generate fewer constraints and variables. However, it is worth noting that such reductions may actually have a negative impact on the overall MILP-solving process. For instance, the recent S-box modeling method proposed by [Udovenko 2021] produces the fewest constraints, but also exhibits the least performance to solve those constraints. (Overall, it is significantly less efficient than some alternatives.) Our findings shed light on the selection of generation methods among various alternatives for practical applications.

We summarize the main contributions as follows.

- We design a high-level cryptography-specific language EasyBC tailored for block ciphers, enabling further automatic analysis.
- We give generic and extensible approaches for automated resistance evaluation of block ciphers written in EasyBC against differential cryptanalysis.
- We implement and extensively evaluate an open-sourced prototype of EasyBC, which confirms the expressiveness of EasyBC and the effectiveness of our approach.

Structure. The rest of the paper is organized as follows. Section 2 presents the background of block ciphers and differential cryptanalysis. Section 3 introduces EasyBC and an overview of our approach. Section 4 presents three key utilities used in our MILP generation. Section 5 and Section 6 describe the word-wise and bit-wise approach. Section 7 describes the extended bit-wise approach. Section 8 reports the experimental results. We discuss related work in Section 9 and conclude this work in Section 10.

2 BACKGROUND

Throughout this paper, $\mathbb{B}$ denotes the Boolean domain $\{0, 1\}$, and $\mathbb{N}$ denotes the set of non-negative integers. Boolean values are treated as integers in arithmetic computations. Given a vector/array $\vec{x}$, $\vec{x}_i$ denotes the $(i+1)$-th entry. Given a matrix M, M_i denotes the $(i+1)$-th row, and $M_{i,j}$ denotes the $(j+1)$-th entry of M_i. An *m-bitstream* is Boolean vector $\vec{b}$ with m entries. We denote by $\oplus$ the bit-wise XOR operator. $\vec{b}\|\vec{b}' = (b_0, \cdots, b_m, b'_0, \cdots, b'_n)$ is the concatenation of two bitstreams $\vec{b} = (b_0, \cdots, b_m)$ and $\vec{b}' = (b'_0, \cdots, b'_n)$. We denote by $\mathsf{bin}(x)$ the binary representation of an unsigned integer x as a bitstream.

2.1 Block ciphers

Block ciphers are a type of symmetric cryptography, which encrypts and decrypts data in fixed-size (e.g., 64 or 128 bits) blocks using the same key [Bogdanov 2010; Knudsen 1997].

Definition 2.1. A *block cipher* is a function $\mathsf{Enc} : \mathbb{B}^{\mathscr{k}} \times \mathbb{B}^{\mathscr{b}} \longrightarrow \mathbb{B}^{\mathscr{b}}$ such that for every key $K \in \mathbb{B}^{\mathscr{k}}$, $\mathsf{Enc}(K, \cdot)$ is a bijective function, where $\mathscr{b}$ is the block size and $\mathscr{k}$ is the key size.

Intuitively, $\mathsf{Enc}(K, \cdot)$ is a keyed-permutation that maps an input block to an output block, where a block is a $\mathscr{b}$-bitstream, the key K determines which permutation to perform. The input and output of $\mathsf{Enc}(K, \cdot)$ are called *plaintext* and *ciphertext*, respectively. A block cipher Enc is *ideal* if it is defined by assigning a uniformly drawn permutation to each of the $2^{\mathscr{k}}$ keyed-permutations. An ideal block cipher is commonly considered to be computationally secure if the key size $\mathscr{k}$ is large enough since the brute-force attack requires $O(2^{\mathscr{k}})$ time. However, it is extremely difficult to implement an ideal block cipher for practical block sizes (e.g., 64 or 128), as one randomly drawn permutation $\mathsf{Enc}(K, \cdot)$ has to be stored for each given key K.

To be efficient yet strong, modern block ciphers apply several (possibly distinct) keyed permutations, where one keyed permutation is a round and implemented by a round function.

Definition 2.2. A *$\mathscr{r}$-round iterative block cipher* ($\mathscr{r}$-IBC) is a function $\mathsf{Enc} : \mathbb{B}^{\mathscr{k}} \times \mathbb{B}^{\mathscr{b}} \longrightarrow \mathbb{B}^{\mathscr{b}}$ such that for every key $K \in \mathbb{B}^{\mathscr{k}}$,

$$\mathsf{Enc}(K, \cdot) = \mathsf{Enc}_{\mathscr{r}}(K^{\mathscr{r}}, \cdot) \circ \cdots \circ \mathsf{Enc}_1(K^1, \cdot),$$

where $\mathsf{Enc}_i : \mathbb{B}^{\mathscr{k}} \times \mathbb{B}^{\mathscr{b}} \longrightarrow \mathbb{B}^{\mathscr{b}}$ is the i-th round with its subkey K^i for $1 \le i \le \mathscr{r}$, symbol $\circ$ denotes function composition, and the subkeys are generated via a key schedule algorithm $g : \mathbb{B}^{\mathscr{k}} \longrightarrow (\mathbb{B}^{\mathscr{k}})^{\mathscr{r}}$, i.e., $g(K) = (K^1, \cdots, K^{\mathscr{r}})$.

Given a key $K \in \mathbb{B}^{\mathscr{k}}$, $\mathsf{Enc}(K, \cdot)$ is used for encryption and $\mathsf{Dec}(K, \cdot)$ is used for decryption, i.e., $\mathsf{Dec}(K, \cdot) = \mathsf{Enc}_1^{-1}(K^1, \cdot) \circ \cdots \circ \mathsf{Enc}_{\mathscr{r}}^{-1}(K^{\mathscr{r}}, \cdot)$, where K^i is the i-th round subkey in Definition 2.2.

Block ciphers can be built in various ways following iterative cipher schemes. The two most widely used are substitution-permutation networks (SPN) [Kam and Davida 1979] and balanced Feistel networks (BFN) [Nyberg 1996]. Note that BFN is used in the former U.S. encryption standard (DES-Data Encryption Standard) [Fox 2000] and SPN is used in the current one (AES-Advanced Encryption Standard) [Daemen and Rijmen 1999]. A brief introduction of BFN and SPN is given in [Sun et al. 2023, Section A].

In the sequel, we fix a $\mathscr{r}$-IBC $\mathsf{Enc} : \mathbb{B}^{\mathscr{k}} \times \mathbb{B}^{\mathscr{b}} \longrightarrow \mathbb{B}^{\mathscr{b}}$ with rounds $\mathsf{Enc}_1, \cdots, \mathsf{Enc}_{\mathscr{r}}$. We assume that the attacker knows all dethe tails of the encryption and decryption except for the secret key.

2.2 Differential Cryptanalysis

Differential cryptanalysis recovers the secret key by exploiting the fact that the probability of some output differences of rounds in a *non-ideal* block cipher is higher than the expected value (i.e., $2^{-\mathscr{b}}$) for certain input differences [Biham and Shamir 1990]. We review the related concepts below.

Difference and differential. Given two $\mathscr{b}$-bitstreams $X \in \mathbb{B}^{\mathscr{b}}$ and $X' \in \mathbb{B}^{\mathscr{b}}$, their (XOR-) *difference* ΔX is defined by $\Delta X = X \oplus X'$. Note that for any fixed difference ΔX, there are exactly $2^{\mathscr{b}}$ pairs (X, X') such that $X \oplus X' = \Delta X$. A *differential* is defined to be a pair of differences $(\Delta X, \Delta Y)$.

Given a pair of inputs $(X, X') \in \mathbb{B}^n \times \mathbb{B}^n$ for a (deterministic) function $f : \mathbb{B}^n \to \mathbb{B}^m$, the *input difference* of the function f is the difference $\Delta X = X \oplus X'$ of the inputs (X, X'), and the *output difference* of f is the difference $f(X) \oplus f(X')$ of the outputs $(f(X), f(X'))$. Clearly, when the input difference is fixed to be ΔX, the output difference of an input X is $f(X) \oplus f(X \oplus \Delta X)$.

The *probability* $\mathsf{Pr}_f(\Delta X, \Delta Y)$ of a given differential $(\Delta X, \Delta Y)$ for the function f is the proportion of inputs $X \in \mathbb{B}^n$ such that the output difference $f(X) \oplus f(X \oplus \Delta X)$ is equal to ΔY, i.e.,

$$\mathsf{Pr}_f(\Delta X, \Delta Y) = \frac{|\{X \in \mathbb{B}^n \mid f(X) \oplus f(X \oplus \Delta X) = \Delta Y\}|}{2^n}.$$

In particular, for an i-th round $\mathsf{Enc}_i : \mathbb{B}^{\mathscr{k}} \times \mathbb{B}^{\mathscr{b}} \to \mathbb{B}^{\mathscr{b}}$, with fixed subkey K^i, $\mathsf{Pr}_{\mathsf{Enc}_i(K^i,\cdot)}(\Delta X^{i-1}, \Delta X^i)$ is the probability of a differential $(\Delta X^{i-1}, \Delta X^i)$ for the function $\mathsf{Enc}_i(K^i, \cdot)$.

It is worth mentioning that in differential cryptanalysis, a key assumption is that the output difference $\mathsf{Enc}_i(K^i, X^{i-1}) \oplus \mathsf{Enc}_i(K^i, X^{i-1} \oplus \Delta X^{i-1})$ of the i-th round is independent of the subkey K^i for any fixed input X^{i-1} and input difference ΔX^{i-1}. As a result, for clarity, $\mathsf{Pr}_{\mathsf{Enc}_i(K^i,\cdot)}(\cdot)$ is simply written as $\mathsf{Pr}_{\mathsf{Enc}_i}(\cdot)$.

Differential characteristic. An *s-round differential characteristic* is a vector $(\Delta X^0, \cdots, \Delta X^s)$ of differences, where

- ΔX^0 a nonzero input difference to the cipher $\mathsf{Enc} : \mathbb{B}^{\mathscr{k}} \times \mathbb{B}^{\mathscr{b}} \to \mathbb{B}^{\mathscr{b}}$,
- for $1 \le i \le s$, $(\Delta X^{i-1}, \Delta X^i)$ is a differential of the i-th round $\mathsf{Enc}_i(K^i, \cdot)$.

Definition 2.3. The *differential characteristic probability* $\mathsf{Pr}_{\mathsf{Enc}(K,\cdot)}(\Delta X^0, \cdots, \Delta X^s)$ of an s-round differential characteristic $(\Delta X^0, \cdots, \Delta X^s)$ is the proportion of inputs $X^0 \in \mathbb{B}^{\mathscr{b}}$ to the cipher Enc such that the output difference of the i-th round Enc_i is ΔX^i for every $1 \le i \le s$ in the two executions of Enc under the two inputs (K, X^0) and $(K, X^0 \oplus \Delta X^0)$, namely,

$$\mathsf{Pr}_{\mathsf{Enc}(K,\cdot)}(\Delta X^0, \cdots, \Delta X^s) = \frac{\left|\left\{X^0 \in \mathbb{B}^{\mathscr{b}} \mid \forall i.1 \le i \le s.\mathsf{Enc}_{\le i}(X_0) \oplus \mathsf{Enc}_{\le i}(X_0 \oplus \Delta X^0) = \Delta X^i\right\}\right|}{2^{\mathscr{b}}}$$

where $\mathsf{Enc}_{\le i} := \mathsf{Enc}_i(K^i, \cdot) \circ \cdots \circ \mathsf{Enc}_1(K^1, \cdot)$ for $1 \le i \le s$.

Recall that we assumed that the output difference $\mathsf{Enc}_i(K^i, X^{i-1}) \oplus \mathsf{Enc}_i(K^i, X^{i-1} \oplus \Delta X^{i-1})$ of the i-th round is independent upon the subkey K^i for any fixed input X^{i-1} and input difference ΔX^{i-1}. Thus, the probability $\mathsf{Pr}_{\mathsf{Enc}(K,\cdot)}(\Delta X^0, \cdots, \Delta X^s)$ does not depend on the s-round subkey K^s, but depends on the other subkeys $K^1, \cdots, K^{s-1}$. It is known that $\mathsf{Pr}_{\mathsf{Enc}(K,\cdot)}(\Delta X^0, \cdots, \Delta X^s)$ is upper bound by $\prod_{i=1}^s \mathsf{Pr}_{\mathsf{Enc}_i}(\Delta X^{i-1}, \Delta X^i)$ [Heys and Tavares 1996], and they are the same if Enc is a Markov cipher and its round subkeys are independent [Lai et al. 1991].

In a resistant block cipher, for any fixed key K, the probability $\mathsf{Pr}_{\mathsf{Enc}(K,\cdot)}(\Delta X^0, \cdots, \Delta X^s)$ should be small enough for any differential characteristic $(\Delta X^0, \cdots, \Delta X^s)$ if the input X^0 is sampled uniformly. However, in practice, $\mathsf{Pr}_{\mathsf{Enc}(K,\cdot)}(\Delta X^0, \cdots, \Delta X^s)$ may be higher than $2^{-\mathscr{b}}$ based on which an attacker can efficiently recover the subkey K^{s+1}. The differential characteristic $(\widetilde{\Delta X}^0, \cdots, \widetilde{\Delta X}^s)$ is said to be *optimal* if it attains the greatest expected differential characteristic probability among all the s-round differential characteristics. We remark that optimal differential characteristics are commonly assumed to be identical for different keys K during the attack, as the actual key is unknown to the adversary before attacking. In the sequel, for simplicity, $\mathsf{Pr}_{\mathsf{Enc}(K,\cdot)}(\cdot)$ is written as $\mathsf{Pr}_{\mathsf{Enc}}(\cdot)$ and $\mathsf{Pr}_{\mathsf{Enc}}(\widetilde{\Delta X}^0, \cdots, \widetilde{\Delta X}^s)$ is refer to as the *maximum expected differential characteristic probability (*MaxEDCP*)*. The key recovering procedure is given in [Sun et al. 2023, Section B], where the number of plaintexts required to infer the $(s+1)$-round subkey K^{s+1} is proportional to $\frac{1}{\mathsf{Pr}_{\mathsf{Enc}}(\widetilde{\Delta X}^0, \cdots, \widetilde{\Delta X}^s)}$ [Heys 2002], i.e., the reciprocal of the MaxEDCP of s-round differential characteristics. As a result, if one could show that an upper bound of $\mathsf{Pr}_{\mathsf{Enc}}(\widetilde{\Delta X}^0, \cdots, \widetilde{\Delta X}^s)$ is no greater than $2^{-\mathscr{b}}$, one would conclude the resistance of the block cipher against such differential cryptanalysis.

2.3 Active S-box, Differential Distribution Table and Branch Number

We introduce some notions of active S-box, differential distribution table and branch number which will be used in our approach.

Active S-boxes. The number of active S-boxes can be used to upper bound the MaxEDCP. Assume S-boxes are distinct in the cipher Enc.

Definition 2.4. [Heys 2002] Given a key K, an input X^0 and an input difference ΔX^0 to the cipher Enc, an S-box $\mathcal{S}$ is *active* if the two inputs to $\mathcal{S}$ are distinct in the two executions of the cipher Enc under two inputs (K, X^0) and $(K, X^0 \oplus \Delta X^0)$, otherwise it is *inactive*.

We denote by $\mathcal{N}_{\mathtt{diff}}$ the minimum number of the active S-boxes in all the possible pairs of executions. The probability of optimal s-round differential characteristics is bounded from above by $p^{\mathcal{N}_{\mathtt{diff}}}$ [Heys 2002], where p denotes the maximum probability $\mathsf{Pr}_{\mathcal{S}}(\Delta X, \Delta Y)$ among all the nonzero differentials $(\Delta X, \Delta Y)$ for any S-box $\mathcal{S}$ which is active in the s-round differential characteristics.

Differential distribution table. A differential distribution table (DDT) is a data structure to represent the distribution Pr_f of a function f for all possible differentials. It also explicitly expresses the dependency (i.e., feasibility) between input and output differences of the function f.

Definition 2.5. Given a function $f : \mathbb{B}^{n_1} \times \cdots \times \mathbb{B}^{n_i} \to \mathbb{B}^{m_1} \times \cdots \times \mathbb{B}^{m_j}$, its DDT $\mathcal{D}_f$ is table such that for every vector of input differences $(\Delta X^1, \cdots, \Delta X^i)$ and every vector of output differences $(\Delta Y^1, \cdots, \Delta Y^j)$, the entry $\mathcal{D}_f(\Delta X^1, \cdots, \Delta X^i, \Delta Y^1, \cdots, \Delta Y^j)$ gives the number of vectors of inputs $(X^1, \cdots, X^i) \in \mathbb{B}^{n_1} \times \cdots \times \mathbb{B}^{n_i}$ such that

$$f(X^1, \cdots, X^i) \oplus f(X^1 \oplus \Delta X^1, \cdots, X^i \oplus \Delta X^i) = (\Delta Y^1, \cdots, \Delta Y^j).$$

The probability $\mathsf{Pr}_f(\Delta X^1, \cdots, \Delta X^i, \Delta Y^1, \cdots, \Delta Y^j)$ can be deduced from the DDT $\mathcal{D}_f$:

$$\mathsf{Pr}_f(\Delta X^1, \cdots, \Delta X^i, \Delta Y^1, \cdots, \Delta Y^j) = \frac{\mathcal{D}_f(\Delta X^1, \cdots, \Delta X^i, \Delta Y^1, \cdots, \Delta Y^j)}{2^{n_1+\cdots+n_i}}.$$

We say the the vector of input and output differences $(\Delta X^1, \cdots, \Delta X^i, \Delta Y^1, \cdots, \Delta Y^j)$ is *feasible* for f, if the probability $\mathsf{Pr}_f(\Delta X^1, \cdots, \Delta X^i, \Delta Y^1, \cdots, \Delta Y^j)$ is nonzero, otherwise it is *infeasible*. When the input space of the function f is small, its DDT $\mathcal{D}_f$ can be computed by enumeration.

Example 2.6. Consider the AND operation $\wedge : \mathbb{B} \times \mathbb{B} \to \mathbb{B}$. We have that

$$\Delta Y = (X^1 \wedge X^2) \oplus ((X^1 \oplus \Delta X^1) \wedge (X^2 \oplus \Delta X^2)).$$

The DDT $\mathcal{D}_\wedge$ is shown in Table 1, e.g., $\mathsf{Pr}_\wedge(0, 0, 0) = \frac{4}{4} = 1$, and $\mathsf{Pr}_\wedge(\Delta X^1, \Delta X^2, \Delta Y) = \frac{2}{4} = \frac{1}{2}$ if $\Delta X^1 = 1$ or/and $\Delta X^2 = 1$ for any fixed ΔY.

The DDT $\mathcal{D}_\vee$ of the OR operation $\vee : \mathbb{B} \times \mathbb{B} \to \mathbb{B}$ is the same as $\mathcal{D}_\wedge$. We can observe that the input difference $(0, 0)$ cannot lead to the output difference 0 for both the AND and OR operations. □

Branch number. The (differential) branch number of a function is also used to characterize the dependency between input and output differences of the function [Daemen and Rijmen 1999].

Table 1. The DDT $(\mathcal{D}_\wedge, \mathcal{D}_\vee)$ for $\wedge/\vee : \mathbb{B} \times \mathbb{B} \rightarrow \mathbb{B}$.

$(\Delta X^1, \Delta X^2)$	(0,0)	(0,1)	(1,0)	(1,1)
$\Delta Y = 0$	4	2	2	2
$\Delta Y = 1$	0	2	2	2

Table 2. Branch numbers of $+, -, \wedge, \vee, \oplus$.

	$+$	$-$	$\wedge$	$\vee$	$\oplus$
$\mathcal{B}^{\min}_{\text{ww}}$ and $\mathcal{B}^{\min}_{\text{bw}}$	2	2	1	1	2
$\mathcal{B}^{\max}_{\text{ww}}$	3	3	3	3	3
$\mathcal{B}^{\max}_{\text{bw}}$	3n-1	3n-1	3n	3n	2n

Definition 2.7. Given a function $f : \mathbb{B}^{n_1} \times \cdots \times \mathbb{B}^{n_i} \longrightarrow \mathbb{B}^{m_1} \times \cdots \times \mathbb{B}^{m_j}$, its *minimum (resp. maximum) word-wise branch number* $\mathcal{B}^{\min}_{\text{ww}}(f)$ (resp. $\mathcal{B}^{\max}_{\text{ww}}(f)$) is defined as

$$\mathcal{B}^{\max}_{\text{ww}}(f) = \max\left\{\mathsf{cnt}(\Delta X^1, \cdots, \Delta X^i, \Delta Y^1, \cdots, \Delta Y^j) \mid \forall 1 \le \ell \le n.\ X^\ell, \Delta X^\ell \in \mathbb{B}^{n_\ell}.\mathsf{BNCond}(f)\right\}$$
$$\mathcal{B}^{\min}_{\text{ww}}(f) = \min\left\{\mathsf{cnt}(\Delta X^1, \cdots, \Delta X^i, \Delta Y^1, \cdots, \Delta Y^j) \mid \forall 1 \le \ell \le n.\ X^\ell, \Delta X^\ell \in \mathbb{B}^{n_\ell}.\mathsf{BNCond}(f)\right\}$$

where $\mathtt{cnt}(\Delta X^1, \cdots, \Delta X^i, \Delta Y^1, \cdots, \Delta Y^j)$ counts the number of nonzero entries in the vector $(\Delta X^1, \cdots, \Delta X^i, \Delta Y^1, \cdots, \Delta Y^j)$ and the branch-number condition $\mathsf{BNCond}(f)$ is:

$$\mathsf{BNCond}(f) = \left(\begin{array}{l} (\Delta X^1 \neq 0 \vee \cdots \vee \Delta X^i \neq 0) \wedge (Y^1, \cdots, Y^j) = f(X^1, \cdots, X^i) \\ \wedge (Y^1 \oplus \Delta Y^1, \cdots, Y^j \oplus \Delta Y^j) = f(X^1 \oplus \Delta X^1, \cdots, X^i \oplus \Delta X^i) \end{array}\right).$$

Likewise, the *minimum (resp. maximum) bit-wise branch number* $\mathcal{B}^{\min}_{\text{bw}}(f)$ (resp. $\mathcal{B}^{\max}_{\text{bw}}(f)$) of the function f is defined except that $\mathtt{cnt}(\Delta X^1, \cdots, \Delta X^i, \Delta Y^1, \cdots, \Delta Y^j)$ counts the number of 1 bits in the bitstream $\Delta X^1 \| \cdots \| \Delta X^i \| \Delta Y^1 \| \cdots \| \Delta Y^j$, i.e., Hamming weight.

Intuitively, when the input differences of the function f are not all 0 bits (i.e., $\Delta X^1 \neq 0 \vee \cdots \vee \Delta X^i \neq 0$), the number of nonzero entries in any input and output differences $(\Delta X^1, \cdots, \Delta X^i, \Delta Y^1, \cdots, \Delta Y^j)$ of the function f ranges from $\mathcal{B}^{\min}_{\text{ww}}(f)$ to $\mathcal{B}^{\max}_{\text{ww}}(f)$, and the Hamming weight of any bitstream $\Delta X^1 \| \cdots \| \Delta X^i \| \Delta Y^1 \| \cdots \| \Delta Y^j$ ranges from $\mathcal{B}^{\min}_{\text{bw}}(f)$ to $\mathcal{B}^{\max}_{\text{bw}}(f)$.

Example 2.8. Consider the function $f_\oplus : \mathbb{B}^n \times \mathbb{B}^n \longrightarrow \mathbb{B}^n$ such that $f_\oplus(X^1, X^2) = X^1 \oplus X^2$. We have: $\mathcal{B}^{\min}_{\text{ww}}(f_\oplus) = \mathcal{B}^{\max}_{\text{ww}}(f_\oplus) = \mathcal{B}^{\min}_{\text{bw}}(f_\oplus) = 2$ and $\mathcal{B}^{\max}_{\text{bw}}(f_\oplus) = 2n$. From $\mathcal{B}^{\min}_{\text{ww}}(f_\oplus) = 2$, we can deduce that at least two of $(X^1, X^2, X^1 \oplus X^2)$ are nonzero if some of (X^1, X^2) is nonzero.

Similarly, $\mathcal{B}^{\min}_{\text{ww}}(f_\odot), \mathcal{B}^{\max}_{\text{ww}}(f_\odot), \mathcal{B}^{\min}_{\text{bw}}(f_\odot)$ and $\mathcal{B}^{\max}_{\text{bw}}(f_\odot)$ for each bit-wise operation $\odot \in \{+, -, \wedge, \vee\}$ can be defined, whose values are given in Table 2, where for clarity, we denote by $\mathcal{B}^{\min}_{\text{ww},\odot}$, $\mathcal{B}^{\max}_{\text{ww},\odot}$, $\mathcal{B}^{\min}_{\text{bw},\odot}$ and $\mathcal{B}^{\max}_{\text{bw},\odot}$ the corresponding numbers of the bit-wise operation $\odot \in \{+, -, \wedge, \vee, \oplus\}$. Furthermore, for a given matrix M and S-box $\mathcal{S}$, $\mathcal{B}^{\min}_{\text{ww},M}$, $\mathcal{B}^{\max}_{\text{ww},M}$, $\mathcal{B}^{\min}_{\text{bw},M}$, $\mathcal{B}^{\max}_{\text{bw},M}$, $\mathcal{B}^{\min}_{\text{ww},\mathcal{S}}$, $\mathcal{B}^{\max}_{\text{ww},\mathcal{S}}$, $\mathcal{B}^{\min}_{\text{bw},\mathcal{S}}$ and $\mathcal{B}^{\max}_{\text{bw},\mathcal{S}}$ are defined accordingly for the linear transformation $f_M(x) = M * x$ and substitution $f_{\mathcal{S}} = \mathcal{S}(x)$. □

3 THE DESIGN OF EASYBC

EasyBC is a high-level, statically-typed, C-like cryptography-specific language, designed for conveniently describing block ciphers but without complicating the subsequent automated security analysis, so that cryptographers can quickly implement and analyze a block cipher especially during the design phase.

3.1 Syntax

The syntax of EasyBC is given in Figure 1.

Boxes. EasyBC features one standard array type and four cryptography-specific array types decorated by `sbox`, `pbox`, $\mathtt{pbox}_\mathtt{m}$ and `ffm`, respectively, that are commonly used for implementing block ciphers. The cryptography-specific arrays are global and immutable, thus called boxes in this paper. In particular, `sbox` defines an array that acts as a lookup-table based S-box for transforming m input bits to n output bits; `pbox` defines an array for performing permutation; $\mathtt{pbox}_\mathtt{m}$ defines a matrix for linear transformation via matrix-vector product. (Note that `pbox` can be implemented via $\mathtt{pbox}_\mathtt{m}$, we provide both for convenience.) `ffm` describes the finite-field multiplication ($\otimes$) which may vary in block ciphers, thus should be defined by users. An `ffm` box is required *only* for performing

OPERATION	$\odot \in \{+, -, \wedge, \vee, \oplus\}$	STMT S ::= $\tau\ x$	Declaration
OPERATION$_2$	$\star \in \{, -, *, /, \%\}$	$\mid x = e;$	Assignment
WIDTH	s ::= $1 \mid 4 \mid 6 \mid 8 \mid 16 \mid \cdots$	$\mid x[\xi] = e;$	Array put
CONSTANT	n ::= $0 \mid 1 \mid 2 \mid 3 \mid \cdots$	$\mid x = f(e_1, \cdots, e_n);$	Function call
BASE TYPE	τ ::= `uints` $\mid$ `uints`$[n]$ $\mid$ `uint`	$\mid \tau\ x = e;$	Decl-Init
POSITION	ξ ::= $n \mid x \mid \xi \star \xi$	$\mid$ `for` $(x$ `from` n_1 `to` $n_2)\{S^+\}$	Range-for

EXPRESSION e ::= $n \mid x \mid e_1 \odot e_2 \mid \sim e \mid M * e$
$\quad\mid x\langle\cdot e\cdot\rangle \mid x\langle e\rangle \mid$ `View`(e, ξ_1, ξ_2)
$\quad\mid$ `touint`$(e_0, \cdots, e_{s-1}) \mid$ `touint`$(e) \mid e \lll \xi \mid e \ggg \xi \mid e[\xi]$

ROUND_FN rnd::= `r_fn uints`$[n]$ $f($`uint` $r,$ `uints`$[n_1]$ $sk,$ `uints`$[n]$ $p)\{S^+$ `return` $y;\}$

SBOX_FN sbox::= `s_fn uints`$_1$ $f($`uints`$_2$ $x)\{S^+$ `return` $y;\}$

BOX box::= `uints`$[n]$ $x = \{n_0, \cdots, n_m\}$
$\quad\mid$ `sbox uints`$[n]$ $x = \{n_0, \cdots, n_m\} \mid$ `pbox uint`$[n]$ $x = \{n_0, \cdots, n_m\}$
$\quad\mid$ `pbox`$_\mathtt{m}$ `uints`$[n][n]$ $M = \{\ \{n_{0,0}, \cdots, n_{0,m}\}, \cdots, \{n_{m,0}, \cdots, n_{m,m}\}\ \}$
$\quad\mid$ `ffm uints`$[n][n]$ $M = \{\ \{n_{0,0}, \cdots, n_{0,m}\}, \cdots, \{n_{t,0}, \cdots, n_{t,m}\}\ \}$

DEF defs::= box | sbox | rnd | defs defs

PROGRAM P::= `@cipher` name defs `fn uints`$[n]$ $f($`uints`$[n_1]$ $k,$ `uints`$[n]$ $p)\{S^+$ `return` $y;\}$

Fig. 1. Syntax of EASYBC.

linear transformations using `pbox`$_\mathtt{m}$, i.e., evaluating expressions of the form $M * x$, thus cannot be explicitly involved in any statements. In this work, we assume a finite field of characteristic 2, so the finite-field addition is the bit-wise XOR ($\oplus$).

Positions. Position ξ is used to express array indices for array get ($e[\xi]$), array slice (`View`(e, ξ_1, ξ_2)), array put ($x[\xi] = e$) and array left/right-rotation ($e \lll \xi$ and $e \ggg \xi$) via the common operations $\{+, -, *, /, \%\}$, whose values can be statically determined after preprocessing (i.e., independent of inputs). After preprocessing, all the positions ξ will be constants.

Expressions. Expressions are defined as usual, including modular addition (+), modular substitution (−), bit-wise AND ($\wedge$), bit-wise OR ($\vee$), bit-wise NOT ($\sim$) and bit-wise XOR ($\oplus$), as well as common cryptography-specific operations.

$M * e$ is a matrix-vector product using finite-field multiplication ($\otimes$) and addition ($\oplus$), where the matrix M must be defined as an array of type `pbox`$_\mathtt{m}$ `uints`$[n][n]$ and the vector e should be an array of type `uints`$[n]$. $x\langle\cdot e\cdot\rangle$ is provided for performing permutation, where x is a P-box. Similarly, $x\langle e\rangle$ is provided for performing substitution, where x is an S-box. `View`(e, ξ_1, ξ_2) is a slice of the array e starting at the index ξ_1 and ending at the index ξ_2 (inclusive), e.g., `View`$((a, b, c, d), 1, 3)$ is (b, c, d). `touint`$(e_0, \cdots, e_{s-1})$ transforms the s-bitstream $(e_0, \cdots, e_{s-1})$ into an s-bit unsigned integer x such that $\text{bin}(x) = (e_0, \cdots, e_{s-1})$, e.g., `touint`$(1, 1, 0, 1)$ is the 4-bit unsigned integer 13. `touint`(e) is the same as `touint`$(e[0], \cdots, e[n-1])$ for the array e of type `uint`$1[n]$. An array e can be left (resp. right) rotated ξ positions via $e \lll \xi$ (resp. $e \ggg \xi$), which can be seen special length-preserving permutations. $e[\xi]$ is array get, the same as `View`(e, ξ, ξ).

Statements. Statements in EASYBC can be declarations, assignments, array puts, returns and round function calls. Note that EASYBC does not support branching statements (e.g., `if-then-else`), because current EASYBC suffices to implement both encryption and decryption processes of block ciphers while branching statements will make modeling complicated when analyzing security. We will evaluate the expressive capability of EASYBC in Section 8.1.

Statement $\tau\ x = e$ is a syntactic sugar of $\tau\ x; x = e$. Statement `for` $(x$ `from` n_1 `to` $n_2)\{S\}$ is a range-for loop. Note that the range $[n_1, n_2]$ is limited to constants, which suffices to express block ciphers. The range variable x should be typed as `uint`, thus could be used in computing indices ξ.

Functions. EasyBC has three types of functions decorated by `r_fn`, `s_fn` or `fn`. A function decorated by `r_fn` is a round function, whose formal parameters are fixed to be the round number r, subkey sk, and text txt. Note that the subkey sk and input text txt should have the same element type `uint`s but may differ in number of elements, and the input text txt and output of a round function should have the same type `uint`$s[n]$. An `s_fn` function is an alternative way to perform substitution instead of using arrays. Small S-boxes (e.g., 4-bit S-box in PRESENT [Bogdanov et al. 2007]) can be easily expressed as arrays, which can facilitate the follow-up security analysis. However, it would be infeasible to express large S-boxes as arrays (e.g., 64-bit S-box [Beierle et al. 2020]), for which `s_fn` functions can be used. An `fn` function defines the encryption (resp. decryption) process of a block cipher. Its parameters include the key k and plaintext (resp. ciphertext) txt. The function body comprises declarations and round function calls for computing the ciphertext (resp. plaintext).

Programs. A program P in EasyBC consists of a cipher name (decorated by @`cipher`), definitions of global boxes and functions, and a definition of an `fn` function describing the cipher.

In this work, following [Mouha et al. 2011; Wu and Wang 2012; Zhang et al. 2018; Zhou et al. 2019], we consider *single-key* differential cryptanalysis, namely, the key is fixed and has no difference in any pair of executions, thus key schedule algorithms are omitted in EasyBC programs. Nevertheless, EasyBC can be easily extended to *related-key* differential cryptanalysis [Biryukov and Nikolić 2010] where the key may differ in some pairs of executions, by introducing a key schedule function. We leave this as interesting future work.

Core language. The (full) language of EasyBC is designed for conveniently describing block ciphers with rich but redundant constructs. To ease the automation of the subsequent security analysis, we have identified a subset of EasyBC as the core language (cf. Figure 1), i.e., in the places highlighted in yellow, positions ξ and expressions e are limited to constants and variables, respectively; in the places highlighted in grey constructs will not be present (i.e., they are to be eliminated by preprocessing the program written in the full language). A program in the core language is obtained by performing loop unrolling, constant-folding, constant propagation and dead-code elimination, on which type checking and security analysis are performed.

Example 3.1. Figure 2 shows a snippet of the 64-bit block cipher PRESENT in EasyBC. The array `s` is an S-box for substitution, the array `p` is the P-box for permutation. The round function `f1` is invoked during the 1-st to the 31-st round. Given the subkey `sk` and text `t`, `f1`(i,`sk`,`t`) for $1 \leq i \leq 31$ produces the output `rtn` of i-th round. In detail, the input `t` is XORed with the subkey `sk` and results in the array `nt`, then the second range-for loop slices `nt` into 16 arrays via calling `View`, each of which is substituted via the S-box `s`, resulting in the array `s_out`. The array `s_out` is processed by applying the array `p` to perform permutation. Finally, the result `rtn` returned.

Remarks on the design choice of EasyBC. In EasyBC, we introduce high-level constructs `View`, `touint` and permutations (i.e., $x\langle\cdot e\cdot\rangle$ and $M * e$) to ease the implementation of block ciphers. The inputs and outputs of round functions are fixed-size blocks which can be implemented by arrays (e.g., `t` and `rtn` in Figure 2). Typically, a block is to be split into small ones on which a look-up table based S-box (e.g., array in EasyBC) of suitable size is applied (e.g., 4-bit S-box `s` in Figure 2). The outputs of S-boxes will be juxtaposed to form a large block on which permutations are performed via either matrix-vector product or P-boxes (e.g., `p1`$\langle\cdot$`s_out`$\cdot\rangle$ in Figure 2). On the other hand, to ease the automation of the subsequent security analysis, most indices are limited to positions ξ whose values can be statically determined (e.g., $e[\xi]$ and `View`(e, ξ_1, ξ_2)), and thus become constants after preprocessing. The loop-up table based S-boxes (i.e., $x\langle e\rangle$) are an exception as they require a

```
@cipher PRESENT
sbox uint4[16] s = {12,5,6,11,9,0,10,13,3,14,15,8,4,7,1,2};
pbox uint[64] p = {0,16,32,48,1,17,33,...15,31,47,63};
r_fn uint1[64] f1(uint r, uint1[64] sk, uint1[64] t){
    uint1[64] nt;
    for(i from 0 to 63){ nt[i] = t[i] ^ sk[i]; }
    uint1[64] s_out;
    for(i from 0 to 15) {
        uint1[4] temp = View(nt, i*4, i*4+3);
        uint4 sbox_in = touint(temp[0], temp[1], temp[2], temp[3]);
        uint4 sbox_out = s1⟨sbox_in⟩      # substitution via S-box
        s_out[i*4]=sbox_out[0];   s_out[i*4+1]=sbox_out[1];
        s_out[i*4+2]=sbox_out[2]; s_out[i*4+3]=sbox_out[3];
    }
    uint1[64] rtn = p1⟨·s_out·⟩;          # permutation via P-box
    return rtn; }
fn uint1[64] enc(uint1[2048] key, uint1[64] plaintext){
    uint1[64] text= plaintext;
    for(i from 1 to 31){  text=f1(i,View(key,(i-1)*64,i*64-1),text);  }
    ....        # execute the last rounds
    return text; }
```

Fig. 2. Code snippet of the 64-bit block cipher PRESENT in EasyBC.

complicated modeling method. It remains open how to handle generic array access with variable indices although currently there appears no such need to specify block ciphers.

3.2 Operational Semantics

Let $\mathbb{X}$ denote a set of variables. An *(evaluation) context* $\sigma : \mathbb{X} \longrightarrow \bigcup_{i \geq 1} \mathbb{N}^i$ is a mapping from variables to values, where a value can be a (fixed-width) non-negative integer or an array. Let $\sigma[x \mapsto v]$ be the context such that $\sigma[x \mapsto v](y) = v$ if $x = y$, otherwise $\sigma[x \mapsto v](y) = \sigma(y)$.

The evaluation judgement is in the form of

$$\sigma \models e : v$$

meaning that the expression e evaluates to the value v under the evaluation context σ.

The evaluation rules are given in Figure 3 (top-part), most of which are standard. Rule (S-Box) states that $x\langle y \rangle$ is a substitution, namely, the entry of the S-box $\sigma(x)$ at the index $\sigma(y)$. Rule (P-Box$_1$) states that $M * x$ is the matrix-vector product of the matrix M and the array $\sigma(x)$. Rule (P-Box$_2$) states that $x\langle\cdot y \cdot\rangle$ is a permutation of the array $\sigma(y)$ according to the indices given by the P-box $\sigma(x) = (j_0, \cdots, j_{n-1})$, where its entry at the index i is the entry of the array $\sigma(y)$ at the index j_i.

The operational semantics of statements is defined as transition rules of the form

$$(\sigma, S) \Rightarrow \sigma'$$

meaning that the execution of the statement S from the state σ results in the state σ'. For a sequence of statements $S_1; S_2; \cdots S_n;$, $(\sigma_0, S_1; S_2; \cdots S_n;) \Rightarrow^+ \sigma_n$ denotes the transitive transition of $\Rightarrow$, i.e., $(\sigma_0, S_1) \Rightarrow \sigma_1$, $(\sigma_1, S_2) \Rightarrow \sigma_2, \cdots, (\sigma_{n-1}, S_n) \Rightarrow \sigma_n$. The transition rules of EasyBC are listed in Figure 3 (bottom-part), which are standard. We denote by $\sigma_0 S_1 \sigma_1 S_2 \sigma_2 \cdots S_n \sigma_n$ the execution of the program P starting from the state σ_0 and ending at the state σ_n, and $(\sigma_{i-1}, S_i) \Rightarrow \sigma_i$ for $1 \leq i \leq n$.

For an execution of an r-IBC, we have

(1) round functions can only be invoked in the `fn` function,
(2) the first arguments in the invoked round functions are the round numbers $1, 2, 3, \cdots, r$, and

$$\frac{}{\sigma \models n : n}\ (\textsc{Const}) \qquad \frac{\sigma(x) = v}{\sigma \models x : v}\ (\textsc{Var}) \qquad \frac{\sigma(x) = v \quad v' = \sim v}{\sigma \models \sim x : v'}\ (\textsc{Not})$$

$$\frac{\sigma(x_1) = n_1 \quad \sigma(x_2) = n_2 \quad n = n_1 \odot n_2}{\sigma \models x_1 \odot x_2 : n}\ (\textsc{Op}) \qquad \frac{\sigma(x) = (v_0, \cdots, v_{n-1}) \quad \sigma(y) = i}{\sigma \models x\langle y\rangle : v_i}\ (\text{S-}\textsc{Box})$$

$$\frac{\sigma(x) = (v_0, \cdots, v_{n-1}), \quad \vec{v} = \left(\bigoplus_{j=0}^{n-1}(M_{0,j} \otimes v_j), \cdots, \bigoplus_{j=0}^{n-1}(M_{n-1,j} \otimes v_j)\right)}{\sigma \models M * x : \vec{v}}\ (\text{P-}\textsc{Box}_1)$$

$$\frac{\sigma(x) = (j_0, \cdots, j_{n-1}) \quad \sigma(y) = (v_0, \cdots, v_{n-1}) \quad \vec{v} = (v_{j_0}, \cdots, v_{j_{n-1}})}{\sigma \models x\langle\cdot y\cdot\rangle : \vec{v}}\ (\text{P-}\textsc{Box}_2)$$

$$\frac{\sigma(x) = (v_0, \cdots, v_{n-1}) \quad \vec{v} = (v_i, \cdots, v_j)}{\sigma \models \mathtt{View}(x, i, j) : \vec{v}}\ (\textsc{View}) \qquad \frac{\mathrm{bin}(n) = (b_0, \cdots, b_{-1})}{\sigma \models \mathtt{touint}(b_0, \cdots, b_{-1}) : n}\ (\textsc{Touint})$$

$$\frac{}{(\sigma, \tau\ x) \Rightarrow \sigma}\ (\textsc{Decl}) \qquad \frac{\sigma \models e : v \quad \sigma' = \sigma[x \mapsto v]}{(\sigma, x = e) \Rightarrow \sigma'}\ (\textsc{Ass})$$

$$\frac{\sigma(x) = (v_0, \cdots, v_{n-1}) \quad \sigma(y) = v \quad \sigma' = \sigma[x \mapsto (v_0, \cdots, v_{i-1}, v, v_{i+1}, \cdots, v_{n-1})]}{(\sigma, x[i] = y) \Rightarrow \sigma'}\ (\textsc{Arr-Put})$$

$$\frac{\begin{array}{c}\tau_0\ \ f(\tau_1\ x_1, \cdots, \tau_m\ x_m)\{S^+\ \mathtt{return}\ y;\} \quad v_1 = \sigma(y_1), \cdots, v_m = \sigma(y_m) \\ \sigma_{\mathtt{in}} = \sigma[x_1 \mapsto v_1] \cdots [x_m \mapsto v_m] \quad (\sigma_{\mathtt{in}}, S^+) \Rightarrow^+ \sigma_{\mathtt{out}} \quad \sigma_{\mathtt{out}}(y) = v \quad \sigma_{\mathtt{ret}} = \sigma[x \mapsto v]\end{array}}{(\sigma, x = f(y_1, \cdots, y_m)) \Rightarrow \sigma_{\mathtt{in}} \qquad (\sigma_{\mathtt{out}}, \mathtt{return}\ y) \Rightarrow \sigma_{\mathtt{ret}}}\ (\textsc{Call-Ret})$$

Fig. 3. The operational semantics of core EasyBC, where $\odot \in \{+, -, \oplus, \wedge, \vee\}$.

(3) the input and output of i-th round are the third argument and return value of the invoked round function whose first argument is i.

3.3 Type System of Core EasyBC

The type system of core EasyBC is designed to disallow certain kinds of illegal programs and provide type information for security analysis.

EasyBC supports the following types, i.e.,

$$\beta ::= \ \mathtt{uint}s \mid \mathtt{uint} \mid \mathtt{uint}s[n] \mid \mathtt{sbox}\ \mathtt{uint}s[n] \mid \mathtt{pbox}\ \mathtt{uint}[n] \mid \mathtt{pbox_m}\ \mathtt{uint}s[n][n].$$

Here, $\mathtt{uint}s$ is for s-bit unsigned integers, $\mathtt{uint}s[n]$ is for vectors (or arrays) of s-bit unsigned integers, $\mathtt{uint}$ is for unsigned integers, and $\mathtt{uint}[n]$ is for vectors (or arrays) of unsigned integers. Note that $\mathtt{uint}s$ is identical to $\mathtt{uint}1[s]$ and $\mathtt{uint}s[1]$. $\mathtt{uint}$ and $\mathtt{uint}[n]$ are used only when the variables under typing are independent of inputs.

Typing expressions. The typing judgement is of the form of

$$T_g, T_l \vdash e : \beta,$$

where (T_g, T_l) is a typing context, e is an expression under typing, and β is a type. The global environment T_g is a mapping from global variables to their types and from function names to function signatures $(\beta_0, \cdots, \beta_n)$ where β_0 is the return type and $\beta_1, \cdots, \beta_n$ are the types of the formal parameters. The local environment T_l is a mapping from local variables to their types. The typing judgement $T_g, T_l \vdash e : \beta$ is valid if e has type β under the typing context (T_g, T_l).

Figure 4 (top-part) gives typing rules for expressions. Rule (T-Uints) express that a non-negative integer n can be typed as $\mathtt{uint}s$ if $n \leq 2^s - 1$. Rules (T-Var) and (T-Not) are defined as usual. Rule (T-Op) ensures that the two operands and result of the operation $\odot \in \{+, -, \oplus, \wedge, \vee\}$ have the same type $\mathtt{uint}s$. Rule (T-Pbox$_1$) ensures that the array x has suitable type w.r.t. the type of the matrix M for matrix-vector product. Rule (T-Pbox$_2$) requires that the elements in the array $x\langle\cdot y\cdot\rangle$ and the

$$\frac{0 \leq n \leq 2^s - 1}{T_g, T_l \vdash n : \texttt{uint}s}\ (\textsc{T-Uints}) \qquad \frac{\odot \in \{+, -, \oplus, \wedge, \vee\} \qquad T_g, T_l \vdash x_i : \texttt{uint}s \text{ for } i = 1, 2}{T_g, T_l \vdash x_1 \odot x_2 : \texttt{uint}s}\ (\textsc{T-Op})$$

$$\frac{T = (x \in \mathbb{X}_f \ ?\ T_l : T_g)}{T_g, T_l \vdash x : T(x)}\ (\textsc{T-Var}) \qquad \frac{T_g(M) = \texttt{pbox}_{\texttt{m}}\ \texttt{uint}s[n][n] \qquad T_g, T_l \vdash x : \texttt{uint}s[n]}{T_g, T_l \vdash M * x : \texttt{uint}s[n]}\ (\textsc{T-Pbox}_1)$$

$$\frac{T_g, T_l \vdash e : \tau}{T_g, T_l \vdash \sim e : \tau}\ (\textsc{T-Not}) \qquad \frac{T_g(x) = \texttt{pbox uint}[n_1] \qquad T_g, T_l \vdash y : \texttt{uint}s[n_2]}{T_g, T_l \vdash x\langle\!\cdot y \cdot\!\rangle : \texttt{uint}s[n_1]}\ (\textsc{T-Pbox}_2)$$

$$\frac{T_g(x) = \texttt{sbox uint}s_1[n] \qquad T_g, T_l \vdash y : \texttt{uint}s_2 \qquad n \geq 2^{s_2}}{T_g, T_l \vdash x\langle y \rangle : \texttt{uint}s_1}\ (\textsc{T-Sbox})$$

$$\frac{T_g, T_l \vdash x : \texttt{uint}s[n] \qquad 0 \leq n_1 \leq n_2 < n \qquad n' = n_2 - n_1 + 1}{T_g, T_l \vdash \texttt{View}(x, n_1, n_2) : \texttt{uint}s[n']}\ (\textsc{T-View})$$

$$\frac{T_g, T_l \vdash x_i : \texttt{uint}1 \text{ for } 0 \leq i < s}{T_g, T_l \vdash \texttt{touint}(x_0, \ldots, x_{s-1}) : \texttt{uint}s}\ (\textsc{T-Touint})$$

$$\frac{T_l(x) = \tau}{T_g, T_l, f \vdash \tau\ x}\ (\textsc{T-Decl}) \qquad \frac{T_l(x) = \texttt{uint}s[n] \qquad T_g, T_l \vdash y : \texttt{uint}s \qquad 0 \leq i < n}{T_g, T_l, f \vdash x[i] = y}\ (\textsc{T-Arr-Put})$$

$$\frac{\begin{array}{c} T_l(x) = \tau \\ T_g, T_l \vdash e : \tau \end{array}}{T_g, T_l, f \vdash\ x = e}\ (\textsc{T-Ass}) \qquad \frac{\begin{array}{c} T_g(f') = (\tau_0, \cdots, \tau_m) \\ T_g, T_l \vdash x : \tau_0 \qquad T_g, T_l \vdash x_i : \tau_i \text{ for } 1 \leq i \leq m \end{array}}{T_g, T_l, f \vdash x = f'(x_1, \cdots, x_m)}\ (\textsc{T-Call})$$

$$\frac{\begin{array}{c} \ell \in \{\texttt{fn}, \texttt{r\_fn}, \texttt{s\_fn}\} \qquad\qquad T_g(f) = (\tau_0, \cdots, \tau_m) \\ T_g, T_l[p_1 \mapsto \tau_1, \cdots, p_m \mapsto t_m, y \mapsto \tau_0], f \vdash S_i \text{ for } 1 \leq i \leq n \end{array}}{T_g, T_l \vdash \ell\ \tau_0\ \ f(\tau_1\ p_1, \cdots, \tau_m\ p_m)\{S_1; \cdots; S_n;\ \texttt{return}\ y;\ \}}\ (\textsc{T-Fn-Def})$$

Fig. 4. The typing rules of core EasyBC.

operand y have the same type, as the P-box x only specifies the element order for the permutation. Rule (T-Sbox) requires that the S-box has a sufficient number of elements (i.e., $n \geq 2^{s_2}$) and the result $x\langle y \rangle$ has the same type as the elements in the S-box x. Note that both S-boxes and P-boxes do not necessarily preserve the length which may occur, e.g., DES. Rule (T-View) requires that the indices n_1 and n_2 are within the bounds of the array x, moreover, the slice $\texttt{View}(x, n_1, n_2)$ is an array with length $n_2 - n_1 + 1$ and its elements have the same type as the elements in the array x. Rule (T-Toint) requires that all the operands x_i have type $\texttt{uint}1$ and the result has type $\texttt{uint}s$ where s is the number of operands.

Typing statements. The typing judgement of a statement is in the form of

$$T_g, T_l, f \vdash S$$

where (T_g, T_l) is a typing context, S is the statement under typing in the function f. We write $T_g, T_l, f \vdash S$ is valid if S is well-typed.

The typing rules are given in Figure 4 (middle-part). Rule (T-Decl) is defined as usual. Rule (T-Ass) requires that the type of the expression e conforms to the declared type of the variable x. Rule (T-Arr-Put) requires that the index i is within the bounds of the array x and the operand y has the same type as the elements in the array x. Rule (T-Call) requires that the types of actual arguments and return conform to the corresponding function signature $T_g(f')$.

Typing programs. Each program P is typed by iteratively typing each function definition. The program is well-typed if all the function definitions are well-typed. The typing judgement of a function definition fn_def is in the form of

$$T_g, T_l \vdash \texttt{fn\_def},$$

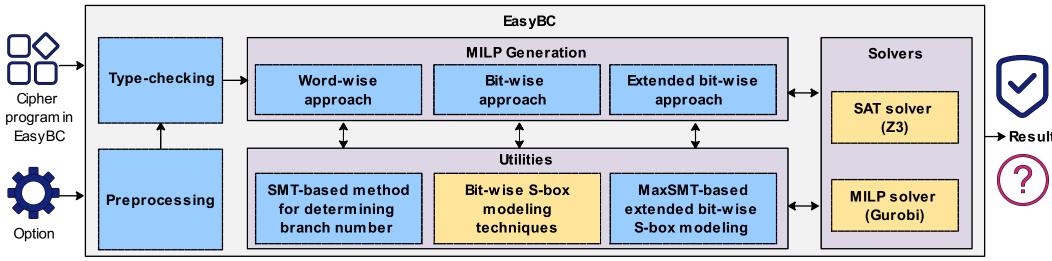

Fig. 5. Overview of our approach.

where (T_g, T_l) is a typing context and `fn_def` is a function definition under typing. The typing judgement $T_g, T_l \vdash$ `fn_def` is valid if the function definition `fn_def` is well-typed. The typing rule (T-Fn-Def) is given in Figure 4 (bottom-part), which enforces the well-typed function body when the formal parameters and return have declared types.

3.4 Compilation

We have implemented an interpreter in C++ for EasyBC to test the operational semantics of programs, making sure that they are consistent with the execution of reference block ciphers. In particular, we compare the output of EasyBC programs with that of running the binary executable compiled from C/C++ programs by GNU C++ compiler (G++). For each cryptographic primitive, we randomly generate inputs and then run the EasyBC program (with our interpreter) and the binary executable. We record their output, as well as the execution time for analysis. The results are given in Section 8.1.

3.5 Overview of Analysis

Recall that we are interested in evaluating the resistance of block ciphers against differential cryptanalysis by bounding the MaxEDCP. A block cipher is considered to be resistant to differential cryptanalysis if MaxEDCP is no greater than $O(2^{\mathcal{b}})$ for the block size $\mathcal{b}$.

Figure 5 gives an overview of our approach. Given a program in full EasyBC together with an option for selecting a particular MILP generation approach and an S-box modeling technique, EasyBC computes an upper bound of the MaxEDCP. The result is conclusive if this upper bound is sufficient to show the resistance of the program. Our approach is not necessarily complete (e.g., in most cases we only compute an upper bound of MaxEDCP), so it may fail to prove the resistance of some programs, although this does not happen in our evaluation (cf. Section 8).

First, the input program is preprocessed to eliminate range-for loops and positional variables by performing loop unrolling, constant-folding, constant propagation and dead-code elimination. The final program will be in the core language of EasyBC. Hereafter, we assume that the given EasyBC program has been preprocessed.

Next, the program is type-checked to disallow certain kinds of illegal programs, e.g., the types of operands in expressions, formal parameters in function definitions and actual arguments in function calls are proper. It also provides type information for security analysis, in particular, the lengths of arrays, the type and the bit widths of array elements, which are used for MILP generation.

After type-checking, we reduce the problem of bounding the MaxEDCP to MILP. The key insight of the reduction is to characterize the dependency (i.e., feasibility) between input and output differences of each operation using integer linear (IL) constraints and bound the MaxEDCP by minimizing an objective function subject to the IL constraints. By utilizing an MILP solver (e.g., Gurobi [Gurobi Optimization 2018]), we can obtain an upper bound of the MaxEDCP. In practice, one may be only interested in proving the resistance against differential cryptanalysis. Hence we also verify whether the MaxEDCP is no greater than a given threshold, an affirmative answer to which would be sufficient to show that the given cipher is resistant against differential cryptanalysis. This strategy is very effective in practice, as few rounds are often sufficient to prove the resistance by leveraging the decomposition approach (cf. Proposition 5.3 and Proposition 7.3).

To this end, we present two different approaches for reducing to MILP. The first one is by determining the lower bound of the minimum number $\mathcal{N}_{\mathtt{diff}}$ of active S-boxes in either word-wise (Section 5) or bit-wise (Section 6) manner, because the MaxEDCP of s-round differential characteristics is bounded from above by $p^{\mathcal{N}_{\mathtt{diff}}}$ [Heys 2002; Sun et al. 2014a], where p denotes the maximum probability $\mathtt{Pr}_{\mathcal{S}}(\Delta X, \Delta Y)$ among all the nonzero differentials $(\Delta X, \Delta Y)$ for any active S-box. Intuitively, the word-wise one models the difference of an s-bitstream under two executions by only one Boolean variable, thus is less involved and produces fewer constraints, but is limited to certain block ciphers e.g., it cannot be directly applied to bit-oriented block ciphers such as PRESENT). In contrast, the bit-wise approach models the difference of each bit by one Boolean variable, thus is more fine-grained and has wider applicability. One may understand that the bit-wise approach implicitly bit-blasts the program and then generates MILP similar to the word-wise approach. The MILP generation in this approach requires the (maximum/minimum word-/bit-wise) branch numbers of some operations and representing S-boxes as IL constraints, for which we propose novel SMT-based methods to automatically determine branch numbers for each operation and implement some recent promising bit-wise S-box modeling techniques.

The first approach is efficient and often effective, but the obtained upper bound may not be sufficiently tight and is not applicable for some ciphers. We provide an extended bit-wise approach to *directly* bound the MaxEDCP (Section 7). In the extended bit-wise approach, the probabilities between input and output differences for each operation are further encoded into IL constraints using additional Boolean variables. This approach may be less efficient but is more accurate than the first approach. We also propose a novel Maximal Satisfiability Modulo Theories (MaxSMT) [Bjørner and Phan 2014] based extended bit-wise S-box modeling method which guarantees that the least number of Boolean variables is used for encoding differential probabilities of S-boxes.

4 UTILITIES

In this section, we present the three key utilities used in our MILP generation.

4.1 SMT-based Method for Determining Branch Numbers

The branch numbers of some operations are required in MILP generation. However, to our best knowledge, existing work usually relies on manual analysis. In this paper, we propose to determine branch numbers of a given operation/function, by reducing to the optimization problem modulo bit-vector theory, which can be solved by off-the-shelf optimizing SMT solver, e.g., Z3 [Bjørner et al. 2015].

Given a function $f : \mathbb{B}^{n_1} \times \cdots \times \mathbb{B}^{n_i} \to \mathbb{B}^{m_1} \times \cdots \times \mathbb{B}^{m_j}$, to compute its minimum (resp. maximum) word-wise branch number $\mathcal{B}_{\mathsf{ww}}^{\min}(f)$ (resp. $\mathcal{B}_{\mathsf{ww}}^{\max}(f)$), by Definition 2.7, we express the condition $\mathsf{BNCond}(f)$ and the additional condition $d = \mathtt{cnt}(\Delta X^1, \cdots, \Delta X^i, \Delta Y^1, \cdots, \Delta Y^j)$ as a quantifier-free SMT formula ϕ_f in the bit-vector theory, and use Z3 to minimize (resp. maximize) the variable d subject to the SMT formula ϕ_f. The optimized value of d is $\mathcal{B}_{\mathsf{ww}}^{\min}(f)$ (resp. $\mathcal{B}_{\mathsf{ww}}^{\max}(f)$). An illustrating example is given in [Sun et al. 2023, Section C.1].

The minimum (resp. maximum) bit-wise branch number $\mathcal{B}_{\mathsf{bw}}^{\min}(f)$ (resp. $\mathcal{B}_{\mathsf{bw}}^{\max}(f)$) can be computed the same as above, except that $\mathtt{cnt}(\Delta X^1, \cdots, \Delta X^i, \Delta Y^1, \cdots, \Delta Y^j)$ counts the number of 1 bits in the bitstream $\Delta X^1 \| \cdots \| \Delta X^i \| \Delta Y^1 \| \cdots \| \Delta Y^j$.

Proposition 4.1. *The minimized (maximized) value of d is*

- $\mathcal{B}^{\min}_{\mathsf{ww}}(f)$ *(resp.* $\mathcal{B}^{\max}_{\mathsf{ww}}(f)$*) when* $\mathtt{cnt}(\Delta X^1, \cdots, \Delta X^i, \Delta Y^1, \cdots, \Delta Y^j)$ *counts the number of nonzero entries in the vector* $(\Delta X^1, \cdots, \Delta X^i, \Delta Y^1, \cdots, \Delta Y^j)$,
- $\mathcal{B}^{\min}_{\mathsf{bw}}(f)$ *(resp.* $\mathcal{B}^{\max}_{\mathsf{bw}}(f)$*) when* $\mathtt{cnt}(\Delta X^1, \cdots, \Delta X^i, \Delta Y^1, \cdots, \Delta Y^j)$ *counts the number of* 1 *bits in the bitstream* $\Delta X^1 \| \cdots \| \Delta X^i \| \Delta Y^1 \| \cdots \| \Delta Y^j$.

4.2 Bit-wise S-box Modeling

Consider an expression $\mathcal{S}\langle X\rangle$ where $\mathcal{S} : \mathbb{B}^n \to \mathbb{B}^m$ is a lookup-table based S-box. It is non-trivial to specify the bit-level dependency of differences between the input X and the result of $\mathcal{S}\langle X\rangle$ (denoted by Y), as the S-box provides only input and output pairs. To resolve this issue, we first compute its DDT $\mathcal{D}_{\mathcal{S}} : \mathbb{B}^n \times \mathbb{B}^m \to \mathbb{N}$ from which IL constraints are generated to characterize the bit-level dependency of differences between X and Y. Recall that for every differential $(\Delta X, \Delta Y) \in \mathbb{B}^n \times \mathbb{B}^m$, $\mathcal{D}_{\mathcal{S}}(\Delta X, \Delta Y)$ gives the number of inputs $X \in \mathbb{B}^n$ such that $\mathcal{S}(X) \oplus \mathcal{S}(X \oplus \Delta X) = \Delta Y$.

Let $\vec{b}$ be the input difference ΔX and $\vec{b}'$ be the output difference ΔY of the S-box $\mathcal{S}$. We have

- $\sum_{i=0}^{n-1} \vec{b}_i = 0 \Rightarrow \sum_{i=0}^{m-1} \vec{b}'_i = 0$, i.e., no output difference if no input difference, namely, if two inputs to $\mathcal{S}$ are the same, then the two outputs are the same. This condition can be characterized by the IL constraint $m \cdot \sum_{i=0}^{n-1} \vec{b}_i \geq \sum_{i=0}^{m-1} \vec{b}'_i$, denoted by $\Psi^1_{\mathcal{S}}$.
- $\sum_{i=0}^{m-1} \vec{b}'_i = 0 \Rightarrow \sum_{i=0}^{n-1} \vec{b}_i = 0$ if $\mathcal{S}$ is injective, i.e., no input difference if no output difference, namely, if two outputs of $\mathcal{S}$ are the same, then the two inputs must be the same. This condition can be exactly characterized by the IL constraint $n \cdot \sum_{i=0}^{m-1} \vec{b}'_i \geq \sum_{i=0}^{n-1} \vec{b}_i$, denoted by $\Psi^2_{\mathcal{S}}$.
- $\mathcal{D}_{\mathcal{S}}(\vec{b}, \vec{b}') \neq 0$, namely, $(\vec{b}, \vec{b}')$ should be feasible for $\mathcal{S}$. We implement and compare the promising techniques [Abdelkhalek et al. 2017; Boura and Coggia 2020; Li and Sun 2022; Sasaki and Todo 2017; Sun et al. 2014b; Udovenko 2021] that can characterize $\mathcal{D}_{\mathcal{S}}(\vec{b}, \vec{b}') \neq 0$ by IL constraints. Hereafter, we denote by $\Psi^3_{\mathcal{S}}$ the set of IL constraints such that $(\vec{b}, \vec{b}')$ is a solution of $\Psi^3_{\mathcal{S}}$ iff $\mathcal{D}_{\mathcal{S}}(\vec{b}, \vec{b}') \neq 0$.

Proposition 4.2. *$(\vec{b}, \vec{b}')$ is feasible input and output differences of $\mathcal{S}$ iff $(\vec{b}, \vec{b}')$ is a solution of $\Psi^3_{\mathcal{S}}$.*

We denote by $\Psi_{\mathcal{S}}$ the set $\Psi^1_{\mathcal{S}} \cup \Psi^2_{\mathcal{S}} \cup \Psi^3_{\mathcal{S}}$ if the S-box $\mathcal{S}$ is injective, otherwise $\Psi^1_{\mathcal{S}} \cup \Psi^3_{\mathcal{S}}$.

4.3 MaxSMT-based Extended Bit-wise S-box Modeling

The above bit-wise S-box modeling method is able to characterize all the feasible differentials $(\Delta X, \Delta Y) \in \mathbb{B}^n \times \mathbb{B}^m$ of the S-box $\mathcal{S}$, but the probability $\mathtt{Pr}_{\mathcal{S}}(\Delta X, \Delta Y) = \frac{\mathcal{D}_{\mathcal{S}}(\Delta X, \Delta Y)}{2^n}$ of differentials is not present in the IL constraints, so it is impossible to bound the MaxEDCP *directly*. We propose a novel MaxSMT-based method which guarantees that the least number of Boolean variables is used for encoding probabilities of differentials.

Definition 4.3. Given two sets of constraints (Φ_1, Φ_2), the MaxSMT problem is to find a solution that satisfies all the constraints in Φ_1 and maximizes the number of satisfied constraints in Φ_2.

Let $V = \{v_1, \cdots, v_h\}$ be the set of nonzero probabilities $\mathtt{Pr}_{\mathcal{S}}(\Delta X, \Delta Y)$ for $(\Delta X, \Delta Y) \in \mathbb{B}^n \times \mathbb{B}^m$. We define the MaxSMT problem $(\Phi^{\mathcal{S}}_1, \Phi^{\mathcal{S}}_2)$ for the S-box $\mathcal{S}$, where

$$\Phi^{\mathcal{S}}_1 = \{\textstyle\sum_{i=1}^{h} c_i \cdot p_{i,j} = -\log_2 v_j \mid 1 \leq j \leq h\} \text{ and } \Phi^{\mathcal{S}}_2 = \{c_1 = 0, \cdots, c_h = 0\},$$

for $1 \leq i, j \leq h$, c_i is a variable over real numbers and $p_{i,j}$ is a Boolean variable. Clearly, for every $1 \leq j \leq h$, $2^{-\sum_{i=1}^{h} c_i \cdot p_{i,j}}$ retains the probability v_j. Since the Boolean variable $p_{i,j}$ can be treated as a variable over real numbers by adding $p_{i,j} = 0 \vee p_{i,j} = 1$ to $\Phi^{\mathcal{S}}_1$, and v_j (hence $\log_2 v_j$) is a constant for each $1 \leq j \leq h$, the problem $(\Phi^{\mathcal{S}}_1, \Phi^{\mathcal{S}}_2)$ is a MaxSMT problem (modulo the theory of real numbers).

A solution of the MaxSMT problem $(\Phi_1^{\mathcal{S}}, \Phi_2^{\mathcal{S}})$ assigns values to the variables c_i's and $p_{i,j}$'s from which the probability v_j for every $1 \le j \le h$ can be obtained. Suppose the solution assigns the values $\{b_{1,j}, \cdots, b_{h,j}\}$ to the Boolean variables $\{p_{1,j}, \cdots, p_{h,j}\}$ and the values $\{t_1, \cdots, t_h\}$ to the variables $\{c_1, \cdots, c_h\}$, we have: $2^{-\sum_{i=1}^{h} t_i \cdot b_{i,j}} = v_j$. Note that $(\Phi_1^{\mathcal{S}}, \Phi_2^{\mathcal{S}})$ is always satisfiable.

We can observe that if $t_i = 0$, the value $b_{i,j}$ of the Boolean variable $p_{i,j}$ for $1 \le j \le h$ can be omitted for retaining all the probabilities in V. We will see later that the values $\{b_{1,j}, \cdots, b_{h,j}\}$ of the Boolean variables $p_{i,j}$'s will be used to encode the probabilities in V, we define $\Phi_2^{\mathcal{S}}$ as $\{c_1 = 0, \cdots, c_h = 0\}$ so that a solution of the MaxSMT problem $(\Phi_1^{\mathcal{S}}, \Phi_2^{\mathcal{S}})$ maximizes the number of 0 bits in the values $\{t_1, \cdots, t_h\}$ of the variables $\{c_1, \cdots, c_h\}$, thus minimizing the number of additional Boolean variables used for encoding the probabilities in V.

Let $\{i_1, \cdots, i_k\}$ be the set of indices of the nonzero values in $\{t_1, \cdots, t_h\}$. To encode all the probabilities in V, we define an extended DDT $\mathcal{D}_{\mathcal{S}}^{\dagger}$ of the S-box $\mathcal{S}$ as follows:

$$\forall(\Delta X, \Delta Y) \in \mathbb{B}^n \times \mathbb{B}^m . 1 \le j \le h.\ \mathcal{D}_{\mathcal{S}}^{\dagger}(\Delta X, \Delta Y, b_{i_1,j}, \cdots, b_{i_k,j}) \ne 0 \text{ iff } \mathsf{Pr}_{\mathcal{S}}(\Delta X, \Delta Y) = v_j.$$

A set $\Psi_{\mathcal{S}}^4$ of constraints over the Boolean variables $\vec{b}, \vec{b'}, p_{i_1}, \cdots, p_{i_k}$ can be generated from the extended DDT $\mathcal{D}_{\mathcal{S}}^{\dagger}$ (cf. Section 4.2) such that

$$\mathcal{D}_{\mathcal{S}}^{\dagger}(\Delta X, \Delta Y, b_{i_1}, \cdots, b_{i_k}) \ne 0 \text{ iff } (\Delta X, \Delta Y, b_{i_1}, \cdots, b_{i_k}) \text{ is a solution of } \Psi_{\mathcal{S}}^4.$$

Proposition 4.4. *For any solution* $(\Delta X, \Delta Y, b_{i_1}, \cdots, b_{i_k})$ *of* $\Psi_{\mathcal{S}}^4$, $\mathsf{Pr}_{\mathcal{S}}(\Delta X, \Delta Y) = 2^{-\sum_{j=1}^{k} t_{i_j} \cdot b_{i_j}}$.

We denote by $\Psi_{\mathcal{S}}^{\dagger}$ the set $\Psi_{\mathcal{S}}^1 \cup \Psi_{\mathcal{S}}^2 \cup \Psi_{\mathcal{S}}^4$ if the S-box $\mathcal{S}$ is injective, otherwise $\Psi_{\mathcal{S}}^1 \cup \Psi_{\mathcal{S}}^4$. An illustrating example is given in [Sun et al. 2023, Section C.2].

5 WORD-WISE APPROACH

In this section, we present an approach for determining the lower bound of the minimum number of active S-boxes by reducing to MILP in a word-wise fashion. It works for programs P where types are `uint`$s[n]$ for a fixed bit size s of the involved S-boxes and individual bits of any entry in an array cannot be changed. Our tool can automatically check if the word-wise approach is applicable. Recall that `uint`s, `uint`$1[s]$ and `uint`$s[1]$ are identical, and we shall use `uint`$s[1]$ hereafter. Note that, in this setting, the program P is free of `touint` expressions.

High-level intuition. Each entry of an array variable x in the program P is modeled as a Boolean variable b, where $b = 1$ in the MILP solution indicates that the entry has a difference when P is executed under two distinct inputs for some fixed key. Thus, a variable of type `uint`$s[n]$ is modeled by a vector $\vec{b}$ of n Boolean variables. Each statement is modeled by a set of IL constraints over the Boolean variables which characterize the propagation of differences through the statement. Furthermore, each S-box $\mathcal{S}$ is associated with a unique Boolean variable $b_{\mathcal{S}}$ such that the S-box $\mathcal{S}$ is active under two execution if $b_{\mathcal{S}} = 1$ (cf. Definition 2.4). The objective function is to minimize the sum of Boolean variables $b_{\mathcal{S}}$ for all the S-boxes $\mathcal{S}$, subject to the extracted IL constraints. The MILP solution gives the lower bound of the minimum number of active S-boxes in the program.

The word-wise MILP generation rules are given by a word-wise differential denotational semantics of EasyBC. We present the denotational semantics for expressions in Section 5.1 and the denotational semantics for statements in Section 5.2.

5.1 Word-wise Differential Denotational Semantics for Expressions

W denotes by $\mathbb{X}_f$ the set of its local variables of a function f, and by $\mathbb{K}_f \subseteq \mathbb{X}_f$ the set of variables that are subkeys. We denote by $|x| = n$ if x has type `uint`$s[n]$.

$[\![\texttt{View}(x,i,j)]\!]^{\mathtt{W}}_{\gamma} = \big(\emptyset, (\vec{b}_i,\cdots,\vec{b}_j)\big)$, where $\vec{b}=\gamma(x)$	$[\![\sim x]\!]^{\mathtt{W}}_{\gamma} = \big(\emptyset, \gamma(x)\big)$
$[\![x_1+x_2]\!]^{\mathtt{W}}_{\gamma} = [\![x_1-x_2]\!]^{\mathtt{W}}_{\gamma} = [\![x_1\oplus x_2]\!]^{\mathtt{W}}_{\gamma} = \big(\Psi^i_{2,3}(b_0,b_1,b_2), b_0\big)$, where $i=1,2$, $b_0=\text{newBV}()$ $\Psi^1_{2,3}(b_0,b_1,b_2)=\big\{b_1+b_2\geq b_0, b_0+b_1\geq b_2,\ b_0+b_2\geq b_1\big\}$ [Li et al. 2019], $b_1=\gamma(x_1)$, $b_2=\gamma(x_2)$ $\Psi^2_{2,3}(b_0,b_1,b_2)=\big\{b'\geq b_0, b'\geq b_1, b'\geq b_2, \sum_{i=0}^{2} b_i\geq 2b'\big\}$ [Mouha et al. 2011], $b'=\text{newBV}()$	
$[\![x_1\wedge x_2]\!]^{\mathtt{W}}_{\gamma} = [\![x_1\vee x_2]\!]^{\mathtt{W}}_{\gamma} = \big(\{b_1+b_2\geq b_0\}, b_0\big)$, where $b_0=\text{newBV}()$, $b_1=\gamma(x_1)$, $b_2=\gamma(x_2)$	
$[\![M*x]\!]^{\mathtt{W}}_{\gamma} = (\Psi^i_M(\vec{b},\vec{b}'),\vec{b}')$, where $i=1,2$, $\vec{b}=\gamma(x)$, $\vec{b}'=\text{newBV}()$, $b''=\text{newBV}()$ $\Psi^1_M(\vec{b},\vec{b}')=\big\{\mathcal{B}^{\min}_{\text{ww},M}\cdot b''\leq\sum_{i=0}^{\lvert x\rvert-1}(\vec{b}_i+\vec{b}'_i)\leq\mathcal{B}^{\max}_{\text{ww},M},\ 2\lvert x\rvert\cdot b''\geq\sum_{i=0}^{\lvert x\rvert-1}(\vec{b}_i+\vec{b}'_i)\big\}$ $\Psi^2_M(\vec{b},\vec{b}')=\big\{\mathcal{B}^{\min}_{\text{ww},M}\cdot b''\leq\sum_{i=0}^{\lvert x\rvert-1}(\vec{b}_i+\vec{b}'_i)\leq\mathcal{B}^{\max}_{\text{ww},M},\ b''\geq\vec{b}_0, b''\geq\vec{b}'_0,\cdots,b''\geq\vec{b}_{\lvert x\rvert-1}, b''\geq\vec{b}'_{\lvert x\rvert-1}\big\}$	
$[\![x\langle\!\cdot y\cdot\!\rangle]\!]^{\mathtt{W}}_{\gamma} = \big(\emptyset, (\vec{b}_{j_0},\cdots,\vec{b}_{j_{n-1}})\big)$, where x is P-box $(j_0,\cdots,j_{n-1})$ and $\vec{b}=\gamma(y)$	
$[\![x\langle y\rangle]\!]^{\mathtt{W}}_{\gamma} = \big(\{b\diamond b'\}, b'\big)$, where $b=\gamma(y)$, $b'=\text{newBV}()$, $\diamond$ is $=$ if x is injective, otherwise $\geq$	

Fig. 6. The word-wise differential denotational semantic rules for expressions.

State. A state γ is a mapping from array entries $(x,i)\in(\mathbb{X}_f\setminus\mathbb{K}_f)\times\mathbb{N}$ to Boolean variables that model the differences of array entries under two executions. For each variable $x\in\mathbb{X}_f\setminus\mathbb{K}_f$,

- $\gamma(x)$ gives the sequence of Boolean variables $\gamma(x,0),\cdots,\gamma(x,\lvert x\rvert-1)$ of the array x.
- $\gamma[(x,i)\mapsto b]$ denotes the update of γ by mapping (x,i) to the Boolean variable b, and $\gamma[x\mapsto\vec{b}]$ denotes the update $\gamma[(x,0)\mapsto\vec{b}_0]\cdots[(x,\lvert x\rvert-1)\mapsto\vec{b}_{\lvert x\rvert-1}]$ for a Boolean vector $\vec{b}$ with $\lvert\vec{b}\rvert=\lvert x\rvert$.

By abuse of notation, $\gamma(x)$ gives the vector $\vec{0}$ with $\lvert\vec{0}\rvert=\lvert x\rvert$ if x is a constant or subkey variable in $\mathbb{K}_f$. Note that $\gamma(x)$ is $\gamma(x,0)$ if x has type $\texttt{uint}s[1]$. We denote by Γ the set of states.

Denotational semantics. The (word-wise differential) denotational semantics of an expression e is given by $[\![e]\!]^{\mathtt{W}}$ that maps each state $\gamma\in\Gamma$ to a pair $(\Psi,\vec{b})$, denoted by $[\![e]\!]^{\mathtt{W}}_{\gamma}$, where

- Ψ is a set of IL constraints over Boolean variables characterizing the dependency/feasiblity of the differences between the support variables and result of e such that the differences is a solution of Ψ iff these differences are feasible for support variables and result of e;
- $\vec{b}$ such that $\lvert\vec{b}\rvert=\lvert e\rvert$ is a vector of the Boolean variables/values that models the difference of the result of e under two executions ($\vec{b}$ may be written as b if $\lvert\vec{b}\rvert=1$ and $\vec{b}_0=b$).

Denotational semantic rules. The semantics for expressions in EasyBC is shown in Figure 6, where the function newBV() returns a fresh Boolean variable b or a vector $\vec{b}$ of fresh Boolean variables according to the context. The semantic rules $[\![\sim x]\!]^{\mathtt{W}}_{\gamma}$, $[\![\texttt{View}(x,i,j)]\!]^{\mathtt{W}}_{\gamma}$ and $[\![x\langle\!\cdot y\cdot\!\rangle]\!]^{\mathtt{W}}_{\gamma}$ are straightforward according to their operational semantics. We explain the others below.

- $[\![x_1\odot x_2]\!]^{\mathtt{W}}_{\gamma}$ for $\odot\in\{+,-,\oplus,\wedge,\vee\}$ gives a pair $\big(\Psi(b_0,b_1,b_2),b_0\big)$, where $\Psi(b_0,b_1,b_2)$ characterizes the dependency of the differences b_0, b_1 and b_2 between $x_1\odot x_2$, x_1 and x_2 according to the maximum and minimum word-wise branch numbers of $\odot$ (cf. Definition 2.7), and $b_0=0$ if $b_1=b_2=0$. For instance, $\mathcal{B}^{\min}_{\text{ww},+}=2$ and $\mathcal{B}^{\max}_{\text{ww},+}=3$ (cf. Table 2), meaning that either $b_0=b_1=b_2=0$ or at least two of them are 1 (i.e., $2\leq b_0+b_1+b_2\leq 3$). For easy reference, these IL constraints are denoted by $\Psi^1_{2,3}$ or $\Psi^2_{2,3}$. Note that the auxiliary Boolean variable b' in $\Psi^2_{2,3}$ is 0 iff $b_0+b_1+b_2=0$.
- $[\![M*x]\!]^{\mathtt{W}}_{\gamma}$ gives the pair $\big(\Psi^i_M(\vec{b},\vec{b}'),\vec{b}'\big)$, where $\Psi^i_M(\vec{b},\vec{b}')$ characterizes the dependency of the differences $\vec{b}$ and $\vec{b}'$ between the entries in the arrays x and $M\odot x$ according to the maximum and minimum word-wise branch numbers of the linear transformation $M\odot x$. Indeed, $\Psi^i_M(\vec{b},\vec{b}')$ enforces that the sum of their input and output differences $\sum_{i=0}^{\lvert x\rvert-1}(\vec{b}_i+\vec{b}'_i)$ either ranges from

$$\frac{}{[\![\tau\ x]\!]^{\mathsf{W}}_{\gamma} = (\emptyset, \gamma, \emptyset)} \qquad \frac{\gamma' = \gamma[(x,i) \mapsto \gamma(y)]}{[\![x[i] = y]\!]^{\mathsf{W}}_{\gamma} = (\emptyset, \gamma', \emptyset)} \qquad \frac{[\![e]\!]^{\mathsf{W}}_{\gamma} = (\Psi, \vec{b}) \quad \gamma' = \gamma[x \mapsto \vec{b}] \quad \Theta = \big(e \text{ is } x'\langle y\rangle ?\ \gamma(y) : \emptyset\big)}{[\![x = e]\!]^{\mathsf{W}}_{\gamma} = (\Psi, \gamma', \Theta)}$$

$$\frac{\begin{array}{c}\tau_0\ \ f(\tau_1\ x_1, \cdots, \tau_m\ x_m)\{S_1; \cdots; S_n; \texttt{return}\ y;\} \qquad \gamma_0 = \gamma[x_1 \mapsto \gamma(y_1)] \cdots [x_m \mapsto \gamma(y_m)] \\ [\![S_1]\!]^{\mathsf{W}}_{\gamma_0} = (\Psi_1, \gamma_1, \Theta_1), \cdots, [\![S_n]\!]^{\mathsf{W}}_{\gamma_{n-1}} = (\Psi_n, \gamma_n, \Theta_n) \quad \Psi = \bigcup_{i=1}^{n} \Psi_i \quad \gamma' = \gamma[x \mapsto \gamma_n(y)] \quad \Theta = \bigcup_{i=1}^{n} \Theta_i\end{array}}{[\![x = g(y_1, \cdots, y_m)]\!]^{\mathsf{W}}_{\gamma} = (\Psi,\ \gamma', \Theta)}$$

Fig. 7. The word-wise differential denotational semantic rules for statements.

$\mathcal{B}^{\min}_{\mathsf{ww},M}$ to $\mathcal{B}^{\max}_{\mathsf{ww},M}$ or is 0 in two alternative ways $\Psi^1_M(\vec{b}, \vec{b'})$ and $\Psi^2_M(\vec{b}, \vec{b'})$, where the auxiliary Boolean variable b'' in an MILP solution is 0 iff $\sum_{i=0}^{|x|-1}(\vec{b}_i + \vec{b}'_i) = 0$. Note that $\mathcal{B}^{\min}_{\mathsf{ww},M} \geq 1$.

- $[\![x\langle y\rangle]\!]^{\mathsf{W}}_{\gamma}$ for S-box x depends upon whether x is injective, which is determined by checking whether some constant in the array appears more than once if the S-box is given by an array, or by checking the satisfiability of the constraint $x \neq x' \wedge f(x) = f(x')$ (via SMT solving) if the S-box is defined by an `s_fn` function f. If it is injective, $[\![x\langle y\rangle]\!]^{\mathsf{W}}_{\gamma}$ gives the pair $\big(\{b = b'\}, b'\big)$, where the Boolean variable b models the difference of y; the fresh Boolean variable b' models the difference of the result $x\langle y\rangle$; and the constraint $b = b'$ ensures that $b = 1$ iff $b' = 1$. If it is non-injective, the constraint $b \geq b'$ is imposed instead of $b = b'$, as $x\langle y\rangle$ *may* differ in two executions *only* if y differs in the two executions.

Lemma 5.1. *Suppose* $[\![e]\!]^{\mathsf{W}}_{\gamma} = (\Psi, \vec{b})$ *with* $\Psi \neq \emptyset$. $\vec{b} = (b_1, \cdots, b_m)$ *is a solution of* Ψ *if and only if* $(b_1, \cdots, b_i)$ *is feasible differences of the operands and result of* e*, where* $(b_{i+1}, \cdots, b_m)$ *is for the possible auxiliary Boolean variables.*

5.2 Word-wise Differential Denotational Semantics for Statements

Denotational semantics for statements. The (word-wise differential) denotational semantics of a statement S is given by $[\![S]\!]^{\mathsf{W}}$ that maps each state $\gamma \in \Gamma$ to a triple (Ψ, γ', Θ), denoted by $[\![S]\!]^{\mathsf{W}}_{\gamma}$, where Ψ is defined as above (i.e., set of IL constraints), γ' is the updated state, and Θ is a set of Boolean variables each of which models the input difference of an S-box under two executions.

Denotational semantic rules. The semantics for statements in EasyBC is shown in Figure 7.

The semantic rules $[\![\tau\ x]\!]^{\mathsf{W}}_{\gamma}$, $[\![x[i] = y]\!]^{\mathsf{W}}_{\gamma}$ and $[\![x = e]\!]^{\mathsf{W}}_{\gamma}$ for declaration $\tau\ x$, array put $x[i] = y$, assignment $x = e$ are straightforward, which updates the state γ accordingly to track the mapping from variables x to Boolean variables $\gamma(x)$ that models the difference of x under two executions, the set of constraints Ψ is collected from the semantics $[\![e]\!]^{\mathsf{W}}_{\gamma}$ of the expression e, and moreover, the Boolean variable $\gamma(y)$ modeling the difference of the input y of an S-box is recorded in Θ.

The semantic rule $[\![x = g(y_1, \cdots, y_m)]\!]^{\mathsf{W}}_{\gamma}$ for a function call $x = g(y_1, \cdots, y_m)$ follows its operational semantics. We first pass the Boolean variables $\gamma(y_i)$ for $1 \leq i \leq m$ that model the differences of the actual arguments y_i to the formal parameters x_i, then iteratively evaluate each statement S_i in its function body, and finally maps the variable x to the Boolean variable $\gamma(y)$ that models the difference of the return y. The set Ψ of IL constraints and the set Θ of Boolean variables modeling the input differences of S-boxes are collected from them of the statements, i.e., Ψ_i's and Θ_i's

5.3 Word-wise Resistance Evaluation

To evaluate the resistance of the program P, we define the semantics $[\![P]\!]^{\mathsf{W}}$ of the program P as the semantics of its `fn` function as follows:

$$[\![P]\!]^{\mathsf{W}} = [\![\texttt{uints}[n]\ f(\texttt{uints}[n_1]\ k,\ \texttt{uints}[n]\ txt)\{S_1; \cdots; S_n; \texttt{return}\ y;\}]\!]^{\mathsf{W}} = (\Psi, \gamma_n, \Theta)$$

where $\Psi = \bigcup_{i=1}^{n} \Psi_i$, $\Theta = \bigcup_{i=1}^{n} \Theta_i$, $[\![S_1]\!]^{\mathsf{W}}_{\gamma_0} = (\Psi_1, \gamma_1, \Theta_1), \cdots, [\![S_n]\!]^{\mathsf{W}}_{\gamma_{n-1}} = (\Psi_n, \gamma_n, \Theta_n)$ and γ_0 is an initial state mapping each array element of the formal parameter txt to a fresh Boolean variable.

Clearly, Φ is the set of IL constraints imposed by the dependency of differences between support variables and results of all the operations in the program P, and $\sum_{b\in\Theta} b$ gives the sum of the number of active S-boxes under two executions of the program P. Determining the lower bound of the minimum number of active S-boxes under two executions of P amounts to minimizing $\sum_{b\in\Theta} b$ subject to $\Phi \cup \{(\sum_{i=0}^{n-1} \gamma(txt, i)) \geq 1\}$, where $(\sum_{i=0}^{n-1} \gamma(txt, i)) \geq 1$ ensures that the input difference of text txt is nonzero, otherwise $\sum_{b\in\Theta} b$ would trivially be 0.

Recall from Section 3.5 that $\mathcal{N}_{\mathtt{diff}}$ denotes the minimum number of active S-boxes in all the possible pairs of executions.

Theorem 5.2. *Let $[\![P]\!]^{\mathsf{W}} = (\Phi, \gamma, \Theta)$ and N be the minimum value of the objective function $\sum_{b\in\Theta} b$ subject to $\Phi \cup \{(\sum_{i=0}^{n-1} \gamma(txt, i)) \geq 1\}$. We have that $N \leq \mathcal{N}_{\mathtt{diff}}$.*

When the MILP program cannot be solved efficiently with large round number s, one can turn to the following decomposition approach.

Proposition 5.3. *Let n_1 and n_2 be the minimum number of the active S-boxes of the first r_1-round and the subsequent r_2-round differential characteristics respectively, then $n_1 + n_2$ is a lower bound of the minimum number of the active S-boxes of the $(r_1 + r_2)$-round differential characteristics.*

In practice, one may be only interested in proving the resistance against differential cryptanalysis instead of computing a bound. In this case, it suffices to prove that $\mathcal{N}_{\mathtt{diff}} \geq \frac{-\mathscr{b}}{\log_2 p}$, where p denotes the maximum probability $\mathtt{Pr}_{\mathcal{S}}(\Delta X, \Delta Y)$ among all the nonzero differentials $(\Delta X, \Delta Y)$ for any S-box $\mathcal{S}$ that is active in s-round differential characteristics and $\mathscr{b}$ is the block size of the cipher. By Theorem 5.2, we only need to verify if $\{\sum_{b\in\Theta} b < \frac{-\mathscr{b}}{\log_2 p}\} \cup \Phi \cup \{(\sum_{i=0}^{n-1} \gamma(txt, i)) \geq 1\}$ is unsatisfiable.

Corollary 5.4. *Let $[\![P]\!]^{\mathsf{W}} = (\Phi, \gamma, \Theta)$. If $\{\sum_{b\in\Theta} b < \frac{-\mathscr{b}}{\log_2 p}\} \cup \Phi \cup \{(\sum_{i=0}^{n-1} \gamma(txt, i)) \geq 1\}$ is unsatisfiable, then the program P with block size $\mathscr{b}$ is resistant against differential cryptanalysis.*

If the s-round cipher P can be partitioned into $\frac{s}{s'}$ identical s'-round ciphers P', by Proposition 5.3 and Corollary 5.4, we can conclude that the cipher P is resistant against differential cryptanalysis if the number of active S-boxes of the cipher P' is no less than $\frac{-s'\cdot\mathscr{b}}{s\cdot\log_2 p}$.

Corollary 5.5. *Let $[\![P']\!]^{\mathsf{W}} = (\Phi, \gamma, \Theta)$. If $\{\sum_{b\in\Theta} b < \frac{-s'\cdot\mathscr{b}}{s\cdot\log_2 p}\} \cup \Phi \cup \{(\sum_{i=0}^{n-1} \gamma(txt, i)) \geq 1\}$ is unsatisfiable, then the program P with block size $\mathscr{b}$ is resistant against differential cryptanalysis.*

6 BIT-WISE APPROACH

In this section, we present a bit-wise approach which, as the word-wise approach, determines the lower bound of the minimum number of active S-boxes, but lifts its limitation requiring that a program uses types $\mathtt{uint}s[n]$ for a fixed bit size s of involved S-boxes and individual bits of any entry in an array cannot be changed.

High-level intuition. Fix a program P, which we normally assume cannot be handled by the word-wise approach. A straightforward idea is to transform the program P to its Boolean counterpart P' by bit-blasting. However, this would introduce a large number of variables and statements, resulting in a prohibitively large MILP problem. In this work, we adopt a strategy to "implicitly" bit-blast, meaning that bit-blasting is performed in the generation of MILP. To this end, each bit of a variable in the program P is modeled by one Boolean variable b, where $b = 1$ in an MILP solution indicates that the corresponding bit differs in P when executed under two distinct inputs for some fixed key. Thus, a variable of type $\mathtt{uint}s[n]$ is modeled by a vector $\vec{b}$ of $s \cdot n$ Boolean variables. The word-wise differential denotational semantics is then lifted to the bit-wise one.

6.1 Bit-wise Differential Denotational Semantics for Expressions

State. We first lift the state γ from word-wise to bit-wise. Let $\|x\| = s \cdot n$ for a variable x of the type $\texttt{uint}s[n]$. A state γ now maps each pair $(x, i) \in (\mathbb{X}_f \setminus \mathbb{K}_f) \times \mathbb{N}$ to a Boolean variable, where

- for each variable x of type $\texttt{uint}s[n]$, $\gamma(x, i \cdot s + j)$ gives a Boolean variable modeling the difference of the $(j+1)$-th most significant bit of the $(i+1)$-th entry x_i in the array x;
- $\gamma(x)$ denotes the sequence $\gamma(x, 0), \gamma(x, 1), \cdots, \gamma(x, \|x\| - 1)$, $\gamma[(x, i) \mapsto b]$ denotes the update of the state γ by mapping (x, i) to the Boolean variable b, and $\gamma[x \mapsto \vec{b}]$ denotes the update $\gamma[(x, 0) \mapsto \vec{b}_0] \cdots [(x, \|x\| - 1) \mapsto \vec{b}_{\|x\|-1}]$ for a Boolean vector $\vec{b}$ with $\|x\| = \|\vec{b}\|$.

Denotational semantics. The (bit-wise differential) denotational semantics of an expression e is given by $[\![e]\!]^{\mathsf{B}}$ that maps each state $\gamma \in \Gamma$ to a pair $(\Psi, \vec{b})$, denoted by $[\![e]\!]^{\mathsf{B}}_\gamma$, where

- Ψ is a set of IL constraints over Boolean variables characterizing the bit-level dependency of differences between the support variables and result of e such that the differences are a solution of Ψ iff these bit-level differences are feasible for support variables and result of e;
- $\vec{b}$ such that $\|\vec{b}\| = \|e\|$ is a Boolean vector modeling the bit-level differences of the result of e.

Denotational semantic rules. The semantics for expressions in EasyBC is shown in Figure 8. The semantic rules $[\![\sim x]\!]^{\mathsf{B}}_\gamma$, $[\![\texttt{View}(x, i, j)]\!]^{\mathsf{B}}_\gamma$, $[\![\texttt{touint}(x_1, \cdots, x_m)]\!]^{\mathsf{B}}_\gamma$ and $[\![x\langle\cdot y\cdot\rangle]\!]^{\mathsf{W}}_\gamma$ are trivial according to their operational semantics. Below, we explain the other non-trivial ones.

- The semantic rule $[\![x \odot y]\!]^{\mathsf{B}}_\gamma$ for $\odot \in \{\wedge, \vee\}$ gives the pair $\left(\{\vec{b}^1_i + \vec{b}^2_i \geq \vec{b}^0_i \mid 0 \leq i < \|x\|\}, \vec{b}^0\right)$, where for every $0 \leq i < \|x\|$, $\vec{b}^1_i + \vec{b}^2_i \geq \vec{b}^0_i$ characterizes that if the $(i+1)$-th bits of the operands x and y have no differences (i.e., $\vec{b}^1_i = \vec{b}^2_i = 0$), then the $(i+1)$-th bit of the result $x \odot y$ has no differences (i.e., $\vec{b}^0_i = 0$). Otherwise, it may have differences (with the probability of $\frac{1}{2}$).
- The semantic rule $[\![x \oplus y]\!]^{\mathsf{B}}_\gamma$ gives the pair $\left(\bigcup_{i=0}^{\|x\|-1} \psi^j_\oplus(\vec{b}^1_i, \vec{b}^2_i, \vec{b}^0_i), \vec{b}^0\right)$, where for each $0 \leq i < \|x\|$, $\psi^j_\oplus(\vec{b}^1_i, \vec{b}^2_i, \vec{b}^0_i)$ characterizes that either the $(i+1)$-th bits of the operands x, y and result $x \oplus y$ have no differences (i.e., $\vec{b}^0_i = \vec{b}^1_i = \vec{b}^2_i = 0$) or exactly two of them have differences (i.e., $\vec{b}^0_i + \vec{b}^1_i + \vec{b}^2_i = 2$).
- The semantic rule $[\![x \odot y]\!]^{\mathsf{B}}_\gamma$ for $\odot \in \{+, -\}$ gives the pair $\left(\bigcup_{i=0}^{\|x\|-1} \Psi_i, \vec{b}^0\right)$, where Ψ_i characterizes the dependency of the differences between the i-th and $(i+1)$-th bits of the operands x, y and result $x \odot y$. The dependency is obtained by bit-blasting $x + y$ via a ripple-carry adder, i.e.,
  - $\mathsf{bin}_i(x + y) = \mathsf{bin}_i(x) \oplus \mathsf{bin}_i(y) \oplus \vec{c}_i$, for every $0 \leq i < \|x\|$;
  - the carry bit $\vec{c}_i = 1$ iff $\mathsf{bin}_{i-1}(x) + \mathsf{bin}_{i-1}(y) + \vec{c}_{i-1} \geq 2$, for every $1 \leq i < \|x\|$, with $\vec{c}_0 = 0$,

  where $\mathsf{bin}_i(x)$ denotes the $(i+1)$-th most significant bit of x. Clearly, the difference $\vec{b}^0_i$ of the bit $\mathsf{bin}_i(x + y)$ depends upon the differences $\vec{b}^1_{i-1}, \vec{b}^2_{i-1}, \vec{b}^1_i$ and $\vec{b}^2_i$ of the bits $\mathsf{bin}_{i-1}(x)$, $\mathsf{bin}_{i-1}(y)$, $\mathsf{bin}_i(x)$ and $\mathsf{bin}_i(y)$. Indeed, we can deduce the dependency:
  - if $\vec{b}^0_{i-1} = \vec{b}^1_{i-1} = \vec{b}^2_{i-1} = 1$, then $\vec{c}_{i-1} \oplus \vec{c}\,'_{i-1} = \vec{b}^3_{i-1} = \vec{b}^3_i = 1$ and $\vec{b}^0_i = \neg(\vec{b}^1_i \oplus \vec{b}^2_i)$,
  - if $\vec{b}^0_{i-1} = \vec{b}^1_{i-1} = \vec{b}^2_{i-1} = 0$, then $\vec{c}_{i-1} \oplus \vec{c}\,'_{i-1} = \vec{b}^3_{i-1} = \vec{b}^3_i = 0$ and $\vec{b}^0_i = \vec{b}^1_i \oplus \vec{b}^2_i$,
  - otherwise $1 \leq \vec{b}^0_{i-1} + \vec{b}^1_{i-1} + \vec{b}^2_{i-1} \leq 2$. Indeed, the probability of $\vec{b}^0_i = \neg(\vec{b}^1_i \oplus \vec{b}^2_i)$, (resp. $\vec{b}^0_i = \vec{b}^1_i \oplus \vec{b}^2_i$ and $\vec{b}^0_i = 1$) is $\frac{1}{2}$.

  The above dependency is characterized by the set of IL constraints Ψ_i. Furthermore, Ψ_0 and Ψ_1 can be simplified by $\vec{c}_0 = 0$. The semantic rule $[\![x - y]\!]^{\mathsf{B}}_\gamma$ is defined the same as $[\![x + y]\!]^{\mathsf{B}}_\gamma$ because $z = x - y$ iff $x = z + y$, and the Boolean variables $\vec{b}^0_i$, $\vec{b}^1_i$ and $\vec{b}^2_i$ in Ψ_i are symmetric.
- The semantic rule $[\![M * x]\!]^{\mathsf{B}}_\gamma$ gives the pair $\left(\bigcup_{i=0}^{|x|-1} \bigcup_{h=0}^{s-1} \Psi^v_{\mathsf{M,i,h}}, \vec{b}'\right)$, where for each $0 \leq i < |x|$ and each $0 \leq h < s$, $\Psi^v_{\mathsf{M,i,h}}$ characterizes the bit-level dependency of the differences between the

$$[\![\sim x]\!]^{\mathsf{B}}_{\gamma} = \big(\emptyset, \gamma(x)\big) \qquad [\![\mathtt{touint}(x_1, \cdots, x_m)]\!]^{\mathsf{B}}_{\gamma} = \big(\emptyset, (\gamma(x_1), \cdots, \gamma(x_m))\big)$$

$$[\![\mathtt{View}(x, i, j)]\!]^{\mathsf{B}}_{\gamma} = \big(\emptyset, (\vec{b}_{i\cdot s+0}, \cdots, \vec{b}_{i\cdot s+s-1}, \cdots, \vec{b}_{j\cdot s+0}, \cdots, \vec{b}_{j\cdot s+s-1})\big), \text{ where } \vec{b} = \gamma(x)$$

$$[\![x \wedge y]\!]^{\mathsf{B}}_{\gamma} = [\![x \vee y]\!]^{\mathsf{B}}_{\gamma} = \big(\{\vec{b}^1_i + \vec{b}^2_i \geq \vec{b}^0_i \mid 0 \leq i < \|x\|\}, \vec{b}^0\big), \text{ where } \vec{b}^1 = \gamma(x),\ \vec{b}^2 = \gamma(y),\ \vec{b}^0 = \mathrm{newBV}()$$

$$[\![x \oplus y]\!]^{\mathsf{B}}_{\gamma} = \big(\textstyle\bigcup_{i=0}^{\|x\|-1} \psi^j_{\oplus}(\vec{b}^1_i, \vec{b}^2_i, \vec{b}^0_i), \vec{b}^0\big), \text{ where } \vec{b}^1 = \gamma(x),\ \vec{b}^2 = \gamma(y),\ \vec{b}^0 = \mathrm{newBV}(), b' = \mathrm{newBV}()$$

$$\psi^1_{\oplus}(b^1, b^2, b^0) = \big\{2 \geq b^0 + b^1 + b^2 \geq 2b',\ b' \geq b^0,\ b' \geq b^1,\ b' \geq b^2\big\} \text{ [Sun et al. 2014a]}$$

$$\psi^2_{\oplus}(b^1, b^2, b^0) = \big\{b^0 + b^1 + b^2 \leq 2,\ b^1 + b^2 \geq b^0,\ b^0 + b^1 \geq b^2,\ b^0 + b^2 \geq b^1\big\} \text{ [Sasaki and Todo 2017]}$$

$$\psi^3_{\oplus}(b^1, b^2, b^0) = \big\{b^0 + b^1 + b^2 = 2b'\big\} \text{ [Cui et al. 2016]}$$

$$[\![x + y]\!]^{\mathsf{B}}_{\gamma} = [\![x - y]\!]^{\mathsf{B}}_{\gamma} = \big(\textstyle\bigcup_{i=0}^{\|x\|-1} \Psi_i, \vec{b}^0\big), \text{ where } \vec{b}^1 = \gamma(x),\ \vec{b}^2 = \gamma(y),\ \vec{b}^0 = \mathrm{newBV}(),$$

$$\Psi_0 = \Psi^i_{\oplus}(\vec{b}^0_0, \vec{b}^1_0, \vec{b}^2_0) \qquad \Psi_1 = \left\{\begin{array}{ll} \vec{b}^0_1 + \vec{b}^1_1 + \vec{b}^2_1 \leq \vec{b}^1_0 + \vec{b}^2_0 + 2, & -\vec{b}^0_1 + \vec{b}^1_1 - \vec{b}^2_1 \leq \vec{b}^1_0 + \vec{b}^2_0, \\ -\vec{b}^0_1 - \vec{b}^1_1 + \vec{b}^2_1 \leq \vec{b}^1_0 + \vec{b}^2_0, & \vec{b}^0_1 - \vec{b}^1_1 - \vec{b}^2_1 \leq \vec{b}^1_0 + \vec{b}^2_0 \end{array}\right\}$$

$$\forall 2 \leq i < \|x\|.\ \Psi_i = \left\{\begin{array}{ll} 4 \geq \sum_{j=0}^2 \vec{b}^j_{i-1} - \vec{b}^0_i + \vec{b}^1_i + \vec{b}^2_i \geq 0, & 4 \geq \sum_{j=0}^2 \vec{b}^j_{i-1} + \vec{b}^0_i + \vec{b}^1_i - \vec{b}^2_i \geq 0, \\ 4 \geq \sum_{j=0}^2 \vec{b}^j_{i-1} + \vec{b}^0_i - \vec{b}^1_i + \vec{b}^2_i \geq 0, & \sum_{j=0}^2 \vec{b}^j_{i-1} + 2 \geq \vec{b}^0_i + \vec{b}^1_i + \vec{b}^2_i \geq \sum_{j=0}^2 \vec{b}^j_{i-1} - 2 \end{array}\right\}$$

$$[\![M * x]\!]^{\mathsf{B}}_{\gamma} = \big(\textstyle\bigcup_{i=0}^{|x|-1} \bigcup_{h=0}^{s-1} \Psi^v_{\mathsf{M,i,h}}, \vec{b}'\big), \text{ where } \vec{b} = \gamma(x), \vec{b}' = \mathrm{newBV}(), b^i_{\mathsf{new}} = \mathrm{newBV}(), v \in \{1, 2\}$$

$$\Psi^v_{\mathsf{M,i,h}} = \psi^v_{\oplus}(b^0, b^1, b^1_{\mathsf{new}}) \cup \psi^v_{\oplus}(b^1_{\mathsf{new}}, b^2, b^2_{\mathsf{new}}) \cup \cdots \cup \psi^v_{\oplus}(b^{m-2}_{\mathsf{new}}, b^{m-1}, b^{m-1}_{\mathsf{new}}) \cup \psi^v_{\oplus}(b^{m-1}_{\mathsf{new}}, b^m, \vec{b}'_{i\cdot s+h})$$

$\Psi^3_{\mathsf{M,i,h}} = \big\{\vec{b}'_{i\cdot s+h} + \sum_{j=0}^m b^j = 2d,\ 0 \leq d \leq \lfloor \frac{m+2}{2} \rfloor\big\}$, where d is a fresh integer variable, $\{b^0, \cdots, b^m\}$ is the set of support variables of $\Big(\bigoplus_{0\leq j<|x|,\ h\leq k<s,\ M_{i,j,k}=1} \vec{b}_{j\cdot s+k}\Big) \oplus \Big(\vec{c}_h \wedge \bigoplus_{0\leq j<|x|,\ 0\leq k<\lfloor \frac{s}{2} \rfloor} \vec{b}_{j\cdot s+2k}\Big)$

$[\![x\langle\cdot y\rangle]\!]^{\mathsf{B}}_{\gamma} = (\emptyset,\ \vec{b}^0 || \cdots || \vec{b}^{|x|-1})$, where $\vec{b} = \gamma(y)$, x is P-box $(j_0, \cdots, j_{n-1})$, and $\vec{b}^i = (\vec{b}_{j_i\cdot s+0}, \cdots, \vec{b}_{j_i\cdot s+s-1})$ for $0 \leq i \leq |x| - 1$

$[\![x\langle y\rangle]\!]^{\mathsf{B}}_{\gamma} = (\Psi_{\mathcal{S}} \cup \Psi^{\mathsf{bn}}_{\mathcal{S}},\ \vec{b}')$, where $\vec{b} = \gamma(y)$, $x : \mathbb{B}^n \to \mathbb{B}^m$ is an S-box $\mathcal{S}$, $\vec{b}' = \mathrm{newBV}()$, $b = \mathrm{newBV}()$, and $\Psi^{\mathsf{bn}}_{\mathcal{S}} = \big\{\mathcal{B}^{\min}_{\mathsf{bw},x} \cdot b \leq \sum_{i=0}^{n-1} \vec{b}_i + \sum_{i=0}^{m-1} \vec{b}'_i \leq \mathcal{B}^{\max}_{\mathsf{bw},x},\ b \geq \vec{b}_i,\ b \geq \vec{b}'_j \mid 0 \leq i < n,\ 0 \leq j < m\big\}$

Fig. 8. The bit-wise differential denotational semantic rules for expressions, where $s = \frac{\|x\|}{|x|}$ denotes the bit width of the entries of the array x, and $\vec{c} = \mathrm{bin}(2 \otimes 2^{s-1})$ is the s-bitstream corresponding to the coefficients of the irreducible polynomial for the underlying finite-field.

array x and the $(h+1)$-th bit $\mathrm{bin}_h(y_i)$ of the $(i+1)$-th entry y_i in the resulting array $y = M * x$. The dependency is obtained by expanding the matrix-vector product as follows:

$$M * x = \Big(\bigoplus_{j=0}^{|x|-1}(M_{0,j} \otimes x_j), \cdots, \bigoplus_{j=0}^{|x|-1}(M_{|x|-1,j} \otimes x_j)\Big).$$

Clearly, the $(i+1)$-th entry y_i is $\bigoplus_{j=0}^{|x|-1}(M_{i,j} \otimes x_j)$ and thus the difference $\vec{b}'_{i\cdot s+h}$ of its $(h+1)$-th bit $\mathrm{bin}_h(y_i)$ is the parity of the differences of the $(h+1)$-th bits in $M_{i,j} \otimes x_j$ for $0 \leq j < h$. The expression $M_{i,j} \otimes x_j$ is bit-blasted by expanding the finite-field multiplication ($\otimes$) using a series of modular left shifts, XOR operations and the coefficients $\vec{c}$ of the irreducible polynomial for the underlying finite field, where $\vec{c} = \mathrm{bin}(2 \otimes 2^{s-1})$. We finally can deduce that the parity of the differences of the $(h+1)$-th bits in $M_{i,j} \otimes x_j$ for $0 \leq j < h$ is

$$\Big(\bigoplus_{0\leq j<|x|,\ h\leq k<s,\ M_{i,j,k}=1} \vec{b}_{j\cdot s+k}\Big) \oplus \Big(\vec{c}_h \wedge \bigoplus_{0\leq j<|x|,\ 0\leq k<\lfloor \frac{s}{2} \rfloor} \vec{b}_{j\cdot s+2k}\Big).$$

Let $\{b^0, \cdots, b^m\}$ be the set of support variables of the above expression. We have:

$$\vec{b}'_{i\cdot s+h} = \bigoplus_{t=0}^m b_t,$$

which can be alternatively characterized by $\Psi^v_{\mathsf{M,i,h}}$ for $v \in \{1, 2, 3\}$.

- The semantic rule $[\![x\langle y\rangle]\!]^{\mathsf{W}}_{\gamma}$ for S-box $x = \mathcal{S}$ gives the pair $(\Psi_{\mathcal{S}} \cup \Psi^{\mathsf{bn}}_{\mathcal{S}}, \vec{b}')$, where $\Psi_{\mathcal{S}}$ is a set of IL constraints characterizing the bit-level dependency of differences between the input y and the result $\mathcal{S}(y)$ (cf. Section 4.2) and $\Psi^{\mathsf{bn}}_{\mathcal{S}}$ enforces that the Hamming weight of the bitstream $\vec{b}\|\vec{b}'$ ranges from $\mathcal{B}^{\min}_{\mathsf{bw},\mathcal{S}}$ to $\mathcal{B}^{\max}_{\mathsf{bw},\mathcal{S}}$. Though $\Psi^{\mathsf{bn}}_{\mathcal{S}}$ is redundant, it often boosts MILP solving.

Lemma 6.1. *Suppose $[\![e]\!]^{\mathsf{B}}_{\gamma} = (\Psi, \vec{b})$ with $\Psi \neq \emptyset$. The assignment $(b_1, \cdots, b_m)$ is a solution of Ψ if and only if $(b_1, \cdots, b_i)$ is feasible bit-level differences of the operands and result of e, where $(b_{i+1}, \cdots, b_m)$ is for the possible auxiliary Boolean variables.*

6.2 Bit-wise Differential Denotational Semantics for Statements

The (bit-wise differential) denotational semantics of a statement S is given by $[\![S]\!]^{\mathsf{B}}$ that maps each state $\gamma \in \Gamma$ to a triple (Ψ, γ', Θ), denoted by $[\![S]\!]^{\mathsf{B}}_{\gamma}$, where Ψ, γ' and Θ are the same as above.

The bit-wise differential semantic rules for statements in EasyBC are the same as those word-wise ones except for array put and S-box access, which are given below:

$$\frac{\vec{b} = \gamma(y) \qquad x \text{ has type } \texttt{uints}[n] \qquad \gamma' = \gamma[(x, i\cdot s + 0) \mapsto \vec{b}_0]\cdots[(x, i\cdot s + s - 1) \mapsto \vec{b}_{s-1}]}{[\![x[i] = y]\!]^{\mathsf{B}}_{\gamma} = (\emptyset, \gamma', \emptyset)}$$

$$\frac{\begin{array}{c}[\![x'\langle y\rangle]\!]^{\mathsf{B}}_{\gamma} = (\Psi, \vec{b}) \qquad \vec{b}' = \gamma(y) \qquad b_x = \mathsf{newBV}() \\ \Psi' = \Psi \cup \{\sum_{j=0}^{\|y\|-1} \vec{b}'_j \geq b_x \geq \vec{b}'_i \mid 0 \leq i < \|y\| \qquad \gamma' = \gamma[x \mapsto \vec{b}]\}\end{array}}{[\![x = x'\langle y\rangle]\!]^{\mathsf{B}}_{\gamma} = (\Psi', \gamma', \{b_x\})}$$

Intuitively, the semantic rule $[\![x[i] = y]\!]^{\mathsf{B}}_{\gamma}$ updates the state γ accordingly by mapping the bits of the $(i+1)$-th entry in the array x to the Boolean variables $\vec{b}$ that model the differences of the bits of the result e. The semantic rule $[\![x = x'\langle y\rangle]\!]^{\mathsf{B}}_{\gamma}$ adds a fresh Boolean variable b_x, where if $b_x = 1$, then the S-box is active, i.e., some bit $\vec{b}'_i$ that models the difference of one bit of input y is nonzero.

6.3 Bit-wise Resistance Evaluation

To evaluate the resistance of the program P in a bit-wise manner, similar to $[\![P]\!]^{\mathsf{W}}$ (cf. Section 5.3), we define the (bit-wise) semantics $[\![P]\!]^{\mathsf{B}}$ of the program P using its `fn` function f as follows.

$$[\![P]\!]^{\mathsf{B}} = [\![\texttt{uints}[n]\ f(\texttt{uints}[n_1]\ k,\ \texttt{uints}[n]\ txt)\{S_1; \cdots; S_n; \texttt{return}\ y;\}]\!]^{\mathsf{B}} = (\Psi, \gamma_n, \Theta)$$

where $\Psi = \bigcup_{i=1}^{n} \Psi_i$, $\Theta = \bigcup_{i=1}^{n} \Theta_i$, $[\![S_1]\!]^{\mathsf{B}}_{\gamma_0} = (\Psi_1, \gamma_1, \Theta_1), \cdots, [\![S_n]\!]^{\mathsf{B}}_{\gamma_{n-1}} = (\Psi_n, \gamma_n, \Theta_n)$ and γ_0 is an initial state mapping each bit of array elements of txt to a fresh Boolean variable. We get that:

Theorem 6.2. *Let $[\![P]\!]^{\mathsf{B}} = (\Phi, \gamma, \Theta)$ and N be the minimum value of the objective function $\sum_{b\in\Theta} b$ subject to the set of IL constraints $\Phi \cup \{\sum_{i=0}^{s\cdot n-1} \gamma(txt, i) \geq 1\}$. We have that $N \leq N_{\texttt{diff}}$.*

Corollary 5.4 and Corollary 5.5 still hold when $[\![P]\!]^{\mathsf{W}} = (\Phi, \gamma, \Theta)$ is replaced by $[\![P]\!]^{\mathsf{B}} = (\Phi, \gamma, \Theta)$.

7 EXTENDED BIT-WISE APPROACH

The lower bound of the minimum number of the active S-boxes is often effective, but it may not be sufficiently tight to prove the resistance [Sun et al. 2014b], as the probabilities between input and output differences for the operations $\wedge, \vee, +, -$ and S-boxes are not fully addressed. Furthermore, the bit-wise approach is not applicable if non-linear layers are implemented in other operations than S-boxes (e.g., SIMON), or S-boxes are too large to be given as arrays (e.g., SPARKLE). In this section, we extend the bit-wise approach to directly bound the MaxEDCP rather than by bounding the minimum number of the active S-boxes.

$$\llbracket x \wedge y \rrbracket^{\mathsf{EB}}_{\gamma} = \llbracket x \vee y \rrbracket^{\mathsf{EB}}_{\gamma} = \big(\bigcup_{i=0}^{\|x\|-1} \Psi_i,\ \vec{b}^0,\ \sum_{i=0}^{\|x\|-1} \vec{p}_i\big), \text{ where } \vec{b}^1 = \Gamma(x),\ \vec{b}^2 = \Gamma(y), \vec{b}^0 = \mathsf{newBV}()$$
$$\Psi_i = \big\{\vec{b}^1_i + \vec{b}^2_i \geq \vec{p}_i,\ \vec{b}^0_i + \vec{b}^1_i + \vec{b}^2_i \leq 3\vec{p}_i\big\}$$

$$\llbracket x + y \rrbracket^{\mathsf{EB}}_{\gamma} = \llbracket x - y \rrbracket^{\mathsf{EB}}_{\gamma} = \big(\bigcup_{i=0}^{\|x\|-1} \Psi_i,\ \vec{b}^0,\ \sum_{i=1}^{\|x\|-1} \vec{p}_i\big), \text{ where } \vec{b}^1 = \Gamma(x),\ \vec{b}^2 = \Gamma(y),\ \vec{b}^0 = \mathsf{newBV}()$$
$$\Psi_0 = \Psi^i_{\oplus}\big(\vec{b}^0_0, \vec{b}^1_0, \vec{b}^2_0\big) \qquad \Psi_1 = \left\{ \begin{array}{c} \vec{b}^0_1 + \vec{b}^1_1 + \vec{b}^2_1 \leq \vec{p}_1 + 2,\ \ -\vec{b}^0_1 + \vec{b}^1_1 - \vec{b}^2_1 \leq \vec{p}_1,\ \ -\vec{b}^0_1 - \vec{b}^1_1 + \vec{b}^2_1 \leq \vec{p}_1, \\ \vec{b}^0_1 - \vec{b}^1_1 - \vec{b}^2_1 \leq \vec{p}_1,\ \ \vec{p}_1 \leq \vec{b}^1_0 + \vec{b}^2_0 \leq 2\vec{p}_1 \end{array} \right\}$$
$$\forall 2 \leq i < \|x\|.\ \Psi_i = \Psi^1_i \cup \Psi^2_i \qquad \Psi^1_i = \left\{ \begin{array}{r} 3 - \vec{p}_i \geq \sum_{j=0}^{2} \vec{b}^j_{i-1} \geq \vec{p}_i,\ \ \vec{p}_i \geq \vec{b}^0_{i-1} - \vec{b}^1_{i-1}, \\ \vec{p}_i \geq \vec{b}^1_{i-1} - \vec{b}^2_{i-1},\ \ \vec{p}_i \geq \vec{b}^2_{i-1} - \vec{b}^0_{i-1} \end{array} \right\}$$
$$\Psi^2_i = \left\{ \begin{array}{ll} 2 - \vec{b}^1_{i-1} + \vec{p}_i \geq -\vec{b}^0_i + \vec{b}^1_i + \vec{b}^2_i \geq -\vec{b}^1_{i-1} - \vec{p}_i, & 2 - \vec{b}^1_{i-1} + \vec{p}_i \geq \vec{b}^0_i + \vec{b}^1_i - \vec{b}^2_i \geq -\vec{b}^1_{i-1} - \vec{p}_i, \\ 2 - \vec{b}^1_{i-1} + \vec{p}_i \geq \vec{b}^0_i - \vec{b}^1_i + \vec{b}^2_i \geq -\vec{b}^1_{i-1} - \vec{p}_i, & 2 + \vec{b}^1_{i-1} + \vec{p}_i \geq \vec{b}^0_i + \vec{b}^1_i + \vec{b}^2_i \geq \vec{b}^1_{i-1} - \vec{p}_i \end{array} \right\}$$

$$\llbracket x\langle y\rangle \rrbracket^{\mathsf{EB}}_{\gamma} = \big(\Psi^{\dagger}_{\mathcal{S}} \cup \Psi^{\mathsf{bn}}_{\mathcal{S}},\ \vec{b}',\ \sum_{j=1}^{k} t_{i_j} \cdot p_{i_j}\big), \text{ where } x \text{ is an S-box } \mathcal{S} : \mathbb{B}^n \to \mathbb{B}^m$$

Fig. 9. The extended bit-wise differential denotational semantic rules for expressions, where $\vec{p} = \mathsf{newBV}()$ is a vector of fresh Boolean variables used to encode probabilities, $\Psi^i_{\oplus}\big(\vec{b}^0_0, \vec{b}^1_0, \vec{b}^2_0\big)$ for $i \in \{1, 2, 3\}$ and $\Psi^{\mathsf{bn}}_{\mathcal{S}}$ are defined in Figure 8, $\Psi^{\dagger}_{\mathcal{S}}$ and $\sum_{j=1}^{k} t_{i_j} \cdot p_{i_j}$ are defined in Section 4.3.

High-level intuition. Apart from the encoding presented in Section 4.3 for S-boxes, we further encode the probabilities between input and output differences for the operations $\wedge, \vee, +, -$ into IL constraints using additional Boolean variables. The weighted sum ϱ of these additional Boolean variables retains the probability $2^{-\varrho}$. Thus, the objective function is designed to minimize ϱ instead of the number of the active S-boxes.

7.1 Extended Bit-wise Differential Denotational Semantics for Expressions

Denotational semantics. The (extended bit-wise differential) denotational semantics of an expression e is given by $\llbracket e \rrbracket^{\mathsf{EB}}$ that maps each state $\gamma \in \Gamma$ to a triple $(\Psi, \vec{b}, \varrho)$, denoted by $\llbracket e \rrbracket^{\mathsf{EB}}_{\gamma}$, where the state γ, set of IL constraint Ψ and Boolean vector $\vec{b}$ are the same as in Section 6.1, and the expression ϱ is a weighted sum of Boolean variables encoding the probability $2^{-\varrho}$.

Denotational semantic rules. The semantic rules $\llbracket e \rrbracket^{\mathsf{EB}}_{\gamma}$ for $\wedge, \vee, +, -$ and S-boxes are shown in Figure 9, otherwise $\llbracket e \rrbracket^{\mathsf{EB}}_{\gamma} = (\Psi, \vec{b}, 0)$ if $\llbracket e \rrbracket^{\mathsf{B}}_{\gamma} = (\Psi, \vec{b})$, meaning that the probability between the input and output differences characterized by Ψ is 1 (i.e., 2^{-0}).

- The semantic rule $\llbracket x \odot y \rrbracket^{\mathsf{EB}}_{\gamma}$ for $\odot \in \{\wedge, \vee\}$ gives the triple $\big(\bigcup_{i=0}^{\|x\|-1} \Psi_i,\ \vec{b}^0,\ \sum_{i=0}^{\|x\|-1} \vec{p}_i\big)$, where for every $0 \leq i < \|x\|$, Ψ_i ensures that the probability of the difference $\vec{b}^0_i$ of the $(i+1)$-th bit in the result $x \odot y$ is $2^{-\vec{p}_i}$ when the differences of the $(i+1)$-th bits of the operands x and y are $\vec{b}^1_i$ and $\vec{b}^2_i$, respectively. Indeed, the probability of $\vec{b}^0_i = 0$ is 2^{-0} when $\vec{b}^1_i = \vec{b}^2_i = 0$, then $\vec{p}_i$ must be 0. The probability of $\vec{b}^0_i = 1$ is 2^{-1} when $\vec{b}^1_i + \vec{b}^2_i \geq 1$, then $\vec{p}_i$ must be 1.
- The semantic rule $\llbracket x \odot y \rrbracket^{\mathsf{EB}}_{\gamma}$ for $\odot \in \{+, -\}$ gives the triple $\big(\bigcup_{i=0}^{\|x\|-1} \Psi_i,\ \vec{b}^0,\ \sum_{i=1}^{\|x\|-1} \vec{p}_i\big)$, where for every $0 \leq i < \|x\|$, Ψ_i ensures that for any fixed $\vec{b}^1_i$ and $\vec{b}^2_i$,
  - if $\vec{b}^0_{i-1} = \vec{b}^1_{i-1} = \vec{b}^2_{i-1}$, $\vec{p}_i = 0$ (i.e., the probability $2^{-\vec{p}_i}$ of $\vec{b}^0_i = \neg(\vec{b}^1_i \oplus \vec{b}^2_i)$ or $\vec{b}^0_i = \vec{b}^1_i \oplus \vec{b}^2_i$ is 1),
  - if $1 \leq \sum_{j=0}^{2} \vec{b}^j_{i-1} \leq 2$, $\vec{p}_i = 1$ (i.e., the probability $2^{-\vec{p}_i}$ of $\vec{b}^0_i = 1$ or $\vec{b}^0_i = 0$ is $\frac{1}{2}$).

  We remark that Ψ_0 and Ψ_1 can be simplified by $\vec{c}_0 = 0$, and the semantic rule $\llbracket x - y \rrbracket^{\mathsf{EB}}_{\gamma}$ is defined the same as $\llbracket x + y \rrbracket^{\mathsf{EB}}_{\gamma}$ because $z = x - y$ iff $x = z + y$.

- The semantic rule $[\![x\langle y\rangle]\!]_\gamma^{\mathsf{EB}}$ follows the result given in Section 4.3, namely, $(\vec{b},\vec{b}',p_{i_1},\cdots,p_{i_k})$ is a solution of $\Psi_{\mathcal{S}}^{\dagger}$ iff the probability $\mathsf{Pr}_{\mathcal{S}}(\vec{b},\vec{b}')$ is $2^{-\sum_{j=1}^{k} t_{i_j}\cdot p_{i_j}}$. Note that t_{i_j}'s are constants and $\Psi_{\mathcal{S}}^{\mathsf{bn}}$ is added to boost MILP solving.

Lemma 7.1. *Suppose $[\![e]\!]_\gamma^{\mathsf{EB}} = (\Psi,\vec{b},\varrho)$ with $\Psi\neq\emptyset$. The assignment $\{b_1,\cdots,b_m,p_1,\cdots,p_n\}$ is a solution of Ψ iff the probability of $\{b_1,\cdots,b_i\}$ being bit-level differences of the operands and result of e is $2^{-\varrho[p_1,\cdots,p_n]}$, where $\varrho[p_1,\cdots,p_n]$ denotes the value of ϱ under the assignment $\{p_1,\cdots,p_n\}$ of the Boolean variables for encoding probabilities, and $\{b_{i+1},\cdots,b_m\}$ is the assignment of the auxiliary Boolean variables if exist.*

7.2 Extended Bit-wise Differential Denotational Semantics for Statements

The (extended bit-wise differential) denotational semantics of a statement S is given by $[\![S]\!]^{\mathsf{EB}}$ that maps each state $\gamma\in\Gamma$ to a triple (Ψ,γ',ϱ), denoted by $[\![S]\!]_\gamma^{\mathsf{EB}}$, where Ψ, γ' and ϱ are the same as above. The extended bit-wise differential semantic rules for statements in EasyBC are similar to those word-wise ones, where $[\![S]\!]_\gamma^{\mathsf{EB}} = (\Psi,\gamma',0)$ if $[\![S]\!]_\gamma^{\mathsf{B}} = (\Psi,\gamma',\emptyset)$, except for

$$\frac{[\![e]\!]_\gamma^{\mathsf{EB}} = (\Psi,\vec{b},\varrho)}{[\![x=e]\!]_\gamma^{\mathsf{EB}} = (\Psi,\gamma[x/\vec{b}],\varrho)}$$

$$\frac{\begin{array}{c}\tau_0\ \ f(\tau_1\ x_1,\cdots,\tau_m\ x_m)\{S_1;\cdots;S_n;\mathtt{return}\ y;\}\qquad \gamma_0=\gamma[x_1/\gamma(y_1)]\cdots[x_m/\gamma(y_m)]\\ [\![S_1]\!]_{\gamma_0}^{\mathsf{EB}}=(\Psi_1,\gamma_1,\varrho_1),\ \cdots,\ [\![S_n]\!]_{\gamma_{n-1}}^{\mathsf{EB}}=(\Psi_n,\gamma_n,\varrho_n)\end{array}}{[\![x=g(y_1,\cdots,y_m)]\!]_\gamma^{\mathsf{EB}} = (\bigcup_{i=1}^n\Psi_i,\ \gamma[x/\gamma_n(y)],\ \sum_{i=1}^n\varrho_i)}$$

Intuitively, the semantic rule $[\![x=g(y_1,\cdots,y_m)]\!]_\gamma^{\mathsf{EB}}$ sums up the expressions ϱ_i's of the statements S_i's in the function body because of $\prod_{i=1}^n 2^{-\varrho_i} = 2^{-\sum_{i=1}^n\varrho_i}$.

7.3 Extended Bit-wise Resistance Evaluation

To evaluate the resistance of the program P in an extended bit-wise manner, we define the (extended bit-wise) semantics $[\![P]\!]^{\mathsf{EB}}$ of the program P using its `fn` function f as follows:

$$[\![P]\!]^{\mathsf{EB}} = [\![\mathtt{uints}[n]\ f(\mathtt{uints}[n_1]\ k,\ \mathtt{uints}[n]\ txt)\{S_1;\cdots;S_n;\mathtt{return}\ y;\}]\!]^{\mathsf{EB}} = (\Psi,\gamma_n,\varrho)$$

where $\Psi=\bigcup_{i=1}^n\Psi_i$, $\varrho=\sum_{i=1}^n\varrho_i$, $[\![S_1]\!]_{\gamma_0}^{\mathsf{B}}=(\Psi_1,\gamma_1,\varrho_1),\cdots,[\![S_n]\!]_{\gamma_{n-1}}^{\mathsf{B}}=(\Psi_n,\gamma_n,\varrho_n)$ and γ_0 is an initial state mapping each bit of array elements of txt to a fresh Boolean variable. We get that:

Theorem 7.2. *Let $[\![P]\!]^{\mathsf{EB}} = (\Phi,\gamma,\varrho)$ and u be the minimum value of ϱ subject to the set of IL constraints $\Phi\cup\{\sum_{i=0}^{s\cdot n-1}\gamma(txt,i)\geq 1\}$. The* MaxEDCP *of the program P is no greater than 2^{-u}.*

Similar to the decomposition approach given in Proposition 5.3, we have

Proposition 7.3. *Let u_1 and u_2 be the* MaxEDCP *of the first s_1-round and the subsequent s_2-round differential characteristics. Then, $u_1\cdot u_2$ is an upper bound of the* MaxEDCP *of (s_1+s_2)-round differential characteristics.*

By Theorem 7.2 and Proposition 7.3, we have the following corollaries.

Corollary 7.4. *Let $[\![P]\!]^{\mathsf{EB}} = (\Phi,\gamma,\varrho)$. If $\{\varrho<\mathscr{b}\}\cup\Phi\cup\{(\sum_{i=0}^{n-1}\gamma(txt,i))\geq 1\}$ is unsatisfiable, then the program P with block size $\mathscr{b}$ is resistant against differential cryptanalysis.*

Corollary 7.5. *If the s-round cipher P can be partitioned into $\frac{s}{s'}$ identical s'-round ciphers P' such that $[\![P']\!]^{\mathsf{EB}} = (\Phi,\gamma,\varrho)$ and $\{\varrho<\frac{s'\cdot\mathscr{b}}{s}\}\cup\Phi\cup\{(\sum_{i=0}^{n-1}\gamma(txt,i))\geq 1\}$ is unsatisfiable, then the program P with block size $\mathscr{b}$ is resistant against differential cryptanalysis.*

Table 3. Statistics of NIST candidates.

Name	EasyBC			C/C++	
	LOC_{bw}	LOC_{ww}	Time	LOC	Time
ASCON(p^a) [Dobraunig et al. 2016]♦	55	N/A	27.7	**42**	1.4
ELEPHANT [Beyne et al. 2020]♦	**37**	N/A	65.3	69	6.6
GIFT-COFB-128 [Banik et al. 2020]	**34**	N/A	23.9	81	3.0
GRAIN(AEAD) [Hell et al. 2021]⋆	**99**	N/A	0.1	146	0.1
ISAP(v2.0) [Dobraunig et al. 2020]♦	**72**	N/A	141.8	94	0.0
PHOTON(Beetle) [Bao et al. 2019]♦	-	**51**	24.4	104	1.8
ROMULUS-128 [Iwata et al. 2020]	**57**	N/A	32.3	113	0.2
SPARKLE256(slim) [Beierle et al. 2019]♦	39	N/A	0.3	**19**	0.0
TinyJAMBU [Wu and Huang 2019]♦	**15**	N/A	143.3	18	0.5
XOODYAK [Daemen et al. 2020]♦	**38**	N/A	21.1	63	0.1

Time is in milliseconds. N/A: word-wise is not applicable. -$\mathscr{B}$ gives the block size required for security evaluation of block ciphers. LOC_{bw}/LOC_{ww}: LOC of bit-/word-wise implementation. ♦ indicates key-less permutations and ⋆ indicates stream cipher where no block size is given.

Table 4. Statistics of other block ciphers.

Name	EasyBC			C/C++	
	LOC_{bw}	LOC_{ww}	Time	LOC	Time
AES-128 [Daemen and Rijmen 1999]	-	**71**	9.0	106	12.5
DES-64 [Fox 2000]	**50**	N/A	4.0	77	2.4
GIFT-64 [Banik et al. 2017]	**34**	N/A	6.5	63	1.9
KLEIN-64 [Gong et al. 2011]	-	**64**	8.1	97	0.5
LBLOCK-64 [Wu and Zhang 2011]	-	**44**	6.0	78	1.7
MIBS-64 [Izadi et al. 2009]	-	**37**	13.9	69	0.3
PICCOLO-64 [Shibutani et al. 2011]	-	**85**	7.4	90	84.5
PRESENT-64 [Bogdanov et al. 2007]	**28**	N/A	8.1	87	0.6
RECTANGLE-64 [Zhang et al. 2015]	**28**	N/A	6.6	80	1.6
SIMON-32 [Beaulieu et al. 2015]	**35**	N/A	3.0	46	0.8
SIMON-48 [Beaulieu et al. 2015]	**35**	N/A	5.9	46	7.1
SKINNY-64 [Beierle et al. 2016]	**38**	N/A	10.7	67	0.3
TWINE-64 [Suzaki et al. 2012]	-	**43**	9.8	61	4.4

8 EVALUATION

Our approach is implemented as an open-source tool. As shown in Figure 5, it utilizes the SMT solver Z3 for computing the branch number and solving MaxSMT problems and Gurobi [Gurobi Optimization 2018] for solving MILP. In general, EasyBC iteratively increases the round number from 1 (cf. Corollary 5.4 and Corollary 7.4). It also partitions an s-round cipher to the maximum number of identical s'-round ciphers and iteratively increases the round number s' (cf. Corollary 5.5 and Corollary 7.5), until the resistance is proved. The tool is designed to be modular and extensible, where each semantic rule has an API wrapper and alternative generation methods can be easily chosen and added. We also incorporate the bounding condition [Matsui 1994; Zhang et al. 2018] into MILP to prune search space which often improves the overall MILP solving. (The detail is given in [Sun et al. 2023, Section E.4].)

All the experiments were conducted on a machine with two Intel Xeon Gold 5118 CPUs (12 cores, 2.30GHz), 64-bit Ubuntu 20.04 LTS, and 128GB RAM. The number of threads for Gurobi is set to 16.

8.1 Expressiveness of EasyBC

To evaluate the expressiveness of EasyBC, we implement 23 realistic cryptographic primitives with EasyBC, consisting of all the 10 finalists of the NIST lightweight cryptography standardization process [NIST 2023] and 13 commonly used block ciphers, covering both SPN ciphers (e.g., AES, PRESENT, and GIFT-COFB) and BFN ciphers (e.g., DES, LBLOCK, TWINE). We stress that we focus on the underlying primitives (i.e., block ciphers and key-less permutations) rather than the full authenticated encryption, message authentication, or hash protocols in the NIST finalists.

The physical source lines of code (LOC [Nguyen et al. 2007]) counted by cloc [Danial 2021] are reported in Tables 3 and 4. We report LOC of word-wise (when available, or otherwise bit-wise) EasyBC implementations. As a comparison, we also report LOC of the C/C++ reference implementations of the NIST finalists and (randomly selected) GitHub open-source C/C++ implementations of other block ciphers. To some extent, a smaller number (highlighted in **bold**) indicates that the language is more succinct in implementing cryptographic primitives.

We observe that EasyBC is sufficiently expressive to easily implement all the 23 cryptographic primitives either in word-wise or bit-wise fashion. More specifically, when cryptographic primitives can be implemented in a word-wise fashion (i.e., PHOTON, AES, KLEIN, LBLOCK, MIBS, PICCOLO, and TWINE), their EasyBC implementations always require considerably less code than the baselines. The main reason is that EasyBC provides high-level constructs of P-box and matrix-vector products for permutations and linear transformations. For cryptographic primitives

Table 5. Results of the word-wise approach, where (n) indicates the number of entire rounds of the cipher and #AS denotes the lower bound of the minimum number of active S-boxes.

Rounds		**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**
AES (14)	**#AS**	1	5	9	**25**	26	30	34	50	51	55	59	75	76	80	N/A										
	Time	0s	0s	0s	**0s**	0s	0s	1s	0s	0s	0s	1s	0s	0s	0s	N/A										
KLEIN (12)	**#AS**	1	5	8	15	16	20	23	30	31	**35**	38	45	N/A												
	Time	1s	0s	0s	0s	0s	0s	0s	0s	0s	**0s**	0s	0s	N/A												
LBLOCK (32)	**#AS**	0	1	2	3	4	6	8	11	14	18	22	24	27	30	**32**	35	36	39	41	44	45	48	50	53	54
	Time	0s	0s	0s	0s	0s	0s	0s	0s	0s	0s	0s	0s	1s	0s	**1s**	1s	1s	3s	3s	2s	6s	5s	62s	62s	80s
MIBS (32)	**#AS**	0	1	2	5	6	7	8	11	12	13	14	17	18	19	20	23	24	25	26	29	30	31	**32**	35	36
	Time	0s	0s	0s	0s	0s	0s	0s	1s	0s	0s	1s	1s	1s	3s	4s	2s	4s	2s	3s	2s	3s	4s	**5s**	4s	7s
PHOTON (12)	**#AS**	1	9	17	81	82	90	98	162	163	171	179	243	N/A												
	Time	0s	0s	0s	0s	0s	0s	0s	0s	0s	0s	1s	0s	N/A												
PICCOLO (25)	**#AS**	0	5	10	15	20	30	**35**	40	45	50	55	60	65	70	75	80	85	90	95	100	105	110	115	120	125
	Time	0s	0s	0s	0s	0s	0s	**0s**	1s	0s	1s	0s	1s	3s	6s	4s	6s	28s	30s	31s	38s	187s	93s	122s	289s	560s
TWINE (36)	**#AS**	0	1	2	3	4	6	8	11	14	18	22	24	27	30	**32**	35	36	39	41	44	45	48	50	53	54
	Time	0s	0s	0s	0s	0s	0s	0s	0s	0s	1s	0s	0s	0s	1s	**0s**	1s	2s	2s	3s	2s	5s	6s	26s	79s	72s

that are implemented in a bit-wise fashion with EasyBC, their EasyBC implementations also require significantly less code than their baselines except for ASCON and SPARKLE, due to the following reasons. (1) One 320-bit block is stored in five 64-bit variables in the reference implementation of ASCON each of which is permuted, while one 320-bit block is stored in one Boolean array in the EasyBC implementation so that the 320-bit Boolean array has to be split before the permutation and merged after permutation. (2) The C++ reference implementation of SPARKLE uses function-like macro definitions and thus is more succinct.

Results of EasyBC **interpreter.** For each realistic cryptographic primitive, we randomly generate 100 inputs to EasyBC programs and binary executables of C/C++ programs. We run the EasyBC program (using our interpreter) and the binary executable for each input and record the output and the execution time. The outputs of the respective programs have been compared with 100% match, which validates the semantics and the interpreter of EasyBC, as well as EasyBC programs. The average execution times over 100 inputs are reported in Tables 3 and 4. While executing via our interpreter is less efficient, it is acceptable for testing EasyBC programs.

8.2 Effectiveness of EasyBC

To evaluate the effectiveness of EasyBC, we first compare the performance of various alternative methods to generate MILP. According to our experiment results (cf. [Sun et al. 2023, Section E]), we select the optimal modeling methods for cryptographic primitives, i.e., $\Psi^1_{2,3}$ and Ψ^1_M are used for modeling the modular addition, substitution, XOR and matrix-vector product respectively in the word-wise approach (cf. Figure 6); $\psi^2_\oplus$, $\Psi^2_{\mathtt{M,i,h}}$ and the technique of [Boura and Coggia 2020, Alg. 2, Alg. 3 and Proposition 3] are used for modeling XOR, matrix-vector product and constructing Ψ^3_S in the bit-wise approach (cf. Figure 8 and Section 4.2); the technique of [Sasaki and Todo 2017] is used for constructing Ψ^4_S in the extended bit-wise approach (cf. Section 4.3). We remark that there is no consensus on the security of key-less permutations yet, and they are used to evaluate the performance of EasyBC instead of security evaluation.

8.2.1 Word-wise Approach. The word-wise approach is evaluated on all the word-wise implementations. The results are reported in Table 5 up to 25 rounds which suffice to prove security. The results for the entire rounds are given in [Sun et al. 2023, Section E.1].

We observe that the execution time (i.e., the MILP solving time) in seconds (s) increases with the round number. The execution time of PICCOLO is considerably higher than that of the others when the round number is large (e.g., ≥13), because PICCOLO generates more constraints. For instance, the number of constraints for the 20-round LBLOCK and TWINE are both 481, while it is

Table 6. Results of the bit-wise approach with the bounding condition, where Timeout is 24 hours.

Rounds		**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
ASCON (12)	**#AS**	1	4	15	N/A											
	Time	0s	1061s	1573s	Timeout											
DES (16)	**#AS**	0	1	2	4	6	8	9	12	12	N/A					
	Time	0s	0s	0s	6s	61s	355s	1073s	27294s	57344s	Timeout					
ELEPHANT (80)	**#AS**	1	2	4	6	10	12	14	16	18	20	22	24	N/A		
	Time	0s	0s	4s	5s	50s	394s	1653s	1592s	2994s	4353s	18294s	20404s	Timeout		
GIFT-COFB (40)	**#AS**	1	2	3	5	7	10	13	17	19	21	N/A				
	Time	0s	0s	1s	7s	7s	38s	775s	2222s	6725s	77870s	Timeout				
GIFT (28)	**#AS**	1	2	3	5	7	**10**	13	16	18	20	22	24	26	N/A	
	Time	0s	0s	1s	1s	2s	**3s**	7s	52s	43s	136s	518s	846s	25561s	Timeout	
PRESENT (31)	**#AS**	1	2	**4**	6	10	12	14	16	18	20	22	24	26	28	N/A
	Time	0s	0s	**0s**	1s	6s	8s	15s	65s	95s	539s	1884s	10271s	38907s	50931s	Timeout
RECTANGLE (25)	**#AS**	1	2	3	4	6	**8**	11	13	15	17	19	21	23	25	27
	Time	1s	0s	0s	1s	8s	**21s**	6s	436s	63s	603s	1135s	1221s	2841s	36333s	37860s
ROMULUS (40)	**#AS**	1	2	**5**	8	12	16	26	36	41	N/A					
	Time	0s	0s	**5s**	48s	87s	378s	1526s	17266s	4043s	Timeout					
SKINNY (36)	**#AS**	**1**	2	5	8	12	16	26	36	41	46	51	55	N/A		
	Time	**0s**	0s	1s	2s	15s	24s	78s	967s	2041s	3610s	15502s	26989s	Timeout		
SPARKLE (7)	**#AS**	1	2	5	6	7	N/A									
	Time	2s	44s	1624s	4416s	17592s	Timeout									

641 for the 20-round PICCOLO. However, the large round number is not always necessary, as the security may have been proved with a small round number (see below).

For block ciphers, $p = 2^{-6}$ and $\ell = 128$ for AES; $p = 2^{-2}$ and $\ell = 64$ for KLEIN, LBLOCK, MIBS, PICCOLO and TWINE; $p = 2^{-2}$ and $\ell = 128$ for PHOTON, where the maximum probability p of the involved S-boxes that are arrays is computed by enumeration. By Corollary 5.4, EasyBC proved that AES (resp. KLEIN, LBLOCK, MIBS, PHOTON, PICCOLO and TWINE) is resistant when the round number s is 4 (resp. 10, 15, 23, 4, 7 and 15), as highlighted in **boldface** in Table 5.

8.2.2 Bit-wise Approach. The bit-wise approach is evaluated on all bit-wise implementations with S-boxes. (The word-wise implementations are excluded.) The results are given in Table 6 up to 15 rounds within 24 hours, where the execution time is the MILP solving time.

Unsurprisingly, we observe that the execution time increases very quickly with the round number due to the blow-up of constraints. For instance, the numbers of constraints and involved variables of ASCON are 6,913 and 1,408 respectively for 1 round, but become 13,825 and 2,496 (resp. 20,737 and 3,584) for 2 (resp. 3) rounds.

For block ciphers, $p = 2^{-2}$ and $\ell = 64$ for DES, PRESENT, RECTANGLE and SKINNY; $p = 2^{-1.415}$ and $\ell = 128$ for GIFT-COFB; $p = 2^{-1.415}$ and $\ell = 64$ for GIFT; and $p = 2^{-2}$ and $\ell = 128$ for ROMULUS. We found that by Corollary 5.4, EasyBC cannot prove that DES (resp. GIFT-COFB, GIFT, PRESENT, RECTANGLE, ROMULUS and SKINNY) is resistant using the lower bounds of numbers of active S-boxes reported in Table 6. Fortunately, by Corollary 5.5, EasyBC proved that GIFT (resp. PRESENT, RECTANGLE, ROMULUS and SKINNY) is resistant with 6 (resp. 3, 6, 3 and 1) rounds, as highlighted in **boldface** in Table 6. Note that the resistance of GIFT-COFB cannot be proved here which will be done by applying the extended bit-wise approach later, while DES is indeed vulnerable to differential cryptanalysis [Biham and Shamir 1990] and EasyBC can return the differential characteristics up to 9 rounds within 24-hour time limit.

We also compared the MILP solving time with/without bounding conditions. Adding the bounding condition often (5 out of 8) improves the efficiency by 1 to 3 times, but does not necessarily improve (and sometimes even worsens) the efficiency (3 out of 8). The results without the bounding condition are given in [Sun et al. 2023, Section E.5].

8.2.3 Extended Bit-wise Approach. The capability of the extended bit-wise approach is evaluated on all the bit-wise implementations of block ciphers and the S-box of SPARKLE (i.e., 4-round Alzette)

Table 7. Results of the extended bit-wise approach with the bounding condition, where Pr denotes the upper bound of the probability of optimal differential characteristics, and Timeout is 24 hours.

Rounds *r*		**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
GIFT-COFB (40)	Pr	$2^{-1.415}$	$2^{-3.415}$	2^{-7}	$2^{-11.415}$	$\mathbf{2^{-17}}$	$2^{-22.415}$	$2^{-28.415}$	N/A							
	Time	0s	6s	9s	124s	**1297s**	5811s	11538s	Timeout							
SIMON-32 (32)	Pr	2^{0}	$\mathbf{2^{-2}}$	2^{-4}	2^{-6}	2^{-8}	2^{-12}	2^{-14}	2^{-18}	2^{-20}	2^{-26}	2^{-30}	2^{-34}	2^{-36}	2^{-38}	2^{-40}
	Time	0s	**0s**	0s	0s	0s	0s	1s	4s	3s	7s	84s	57s	820s	191s	6327s
SIMON-48 (36)	Pr	2^{0}	2^{-2}	$\mathbf{2^{-4}}$	2^{-6}	2^{-8}	2^{-12}	2^{-14}	2^{-18}	2^{-20}	2^{-26}	2^{-30}	2^{-36}	2^{-38}	2^{-44}	2^{-46}
	Time	0s	0s	**1s**	0s	0s	1s	3s	5s	14s	16s	101s	95s	402s	505s	53920s
Alzette (12)	Pr	2^{0}	2^{-1}	2^{-2}	2^{-6}	2^{-10}	2^{-18}	N/A								
	Time	0s	1s	2s	75s	221s	3897s	Timeout								

that cannot be proved or analyzed before. The results are given in Table 7 up to 15 rounds within 24 hours, where the execution time is the MILP solving time.

Unsurprisingly, the execution time of the extended bit-wise approach is longer than that of the bit-wise approach. For instance, on the 7-round GIFT-COFB, the execution time of the extended bit-wise approach is 11,538s while the execution time of the bit-wise approach is 1,074s. This is because probabilities are explicitly encoded using additional Boolean variables in the extended bit-wise approach, resulting in more difficult MILP instances.

Note the block size: GIFT-COFB $\ell = 128$, SIMON-32 $\ell = 32$, and SIMON-48 $\ell = 48$. By Corollary 7.5, EasyBC prove that GIFT-COFB (resp. SIMON-32 and SIMON-48) is resistant with 5 (resp. 2 and 3) rounds.

9 RELATED WORK

[Matsui 1994] proposed the first algorithm for automated resistance analysis of block ciphers against differential cryptanalysis. To further enhance efficiency, various heuristics have been proposed to reduce the search space [Aoki et al. 1997; Bao et al. 2014; Biryukov and Nikolić 2010; Ji et al. 2021]. However, these heuristics generally rely on cipher-specific optimizations, necessitating sophisticated programming skills. Additionally, creating highly reusable code that can be easily adapted for different ciphers is non-trivial [Zhang et al. 2018].

In recent years, a more promising approach based on MILP has been developed. [Mouha et al. 2011] proposed to determine the lower bound of the minimum number of active S-boxes via MILP solving. As an early attempt, it only considered the word-wise modeling for the XOR operation, S-box, and linear transformation. To partially lift this limitation, [Sun et al. 2013] introduced the bit-wise modeling. To precisely characterize S-boxes in MILP, [Sun et al. 2014a] proposed to construct IL constraints from the H-representation of the S-box DDT and reduced the number of constraints via a greedy algorithm. Later, Sun *et al.* [Sun et al. 2014b] proposed a bit-wise modeling method for the AND operation and extended the method of [Sun et al. 2014a] to directly bound the MaxEDCP by encoding the probabilities between input and output differences of S-boxes in IL constraints.

Since then, plenty of modeling methods for specific operations have been proposed, aimed at improving efficiency and applicability. [Sasaki and Todo 2017] proposed an MILP-based algorithm to reduce the number of constraints obtained from the H-representation of the S-box DDT. [Abdelkhalek et al. 2017] proposed to construct IL constraints from the minimized product-of-sum representation of S-box DDT instead of the H-representation. Along this line, [Boura and Coggia 2020; Li and Sun 2022; Sun 2021; Udovenko 2021] generate more diverse constraints from the S-box DDT, allowing the number of the resulting constraints to be further reduced by applying greedy or MILP-based algorithms; [Cui et al. 2016; Li et al. 2019; Sasaki and Todo 2017; Yin et al. 2017] proposed a new modeling for the XOR operation; [Boura and Coggia 2020; Ilter and Selçuk 2021; Zhang and Zhang 2018] proposed another modeling for linear transformation; [Fu et al. 2016] proposed a modeling method for the modular addition operation; and [Chen et al. 2015; Liu et al.

2017; Wang et al. 2018] proposed a modeling method for the rotation-AND operation. Besides new modeling methods for specific operations, optimization strategies have also been proposed. [Zhang et al. 2018] incorporated the bounding condition of [Matsui 1994] into the MILP-based method; [Zhou et al. 2019] proposed partitioning all possible differential characteristics into subsets, each of which is analyzed by the MILP-based method.

Another direction is to resort to SAT/SMT solving. [Mouha and Preneel 2013] proposed the first SAT/SMT modeling for the XOR, modular addition, and rotation operations. It has been extended to handle a specific permutation [Aumasson et al. 2014], the AND operation and rotation with constants [Kölbl et al. 2015], independent modular addition [Song et al. 2016], S-boxes [Liu et al. 2021; Sun et al. 2018], and modular addition with constants [Azimi et al. 2022]. In contrast to the MILP-based method which determines the lower bound of the minimum number of active S-boxes or the upper bound of MaxEDCP, SAT/SMT-based methods can only verify whether a given number is a bound. Recently, both the bounding condition of [Matsui 1994] and MILP-based method have been combined with the SAT/SMT-based method [Makarim and Rohit 2022; Sun et al. 2021] to improve the efficiency. To facilitate the comparison of all the above MILP/SAT/SMT-based approaches, a summary is given in [Sun et al. 2023, Section F].

Despite significant progress in the field, existing works primarily concentrate on ad-hoc modeling methods designed for specific operations, but do not offer systematic methods for determining word-wise or bit-wise branch numbers and encoding probabilities between input and output differences. There is a lack of language support, unified computational approaches and full automation. This limitation forces cryptanalysts to individually model each cipher within the tool or create a model generation script for every individual cipher, resulting in a complex, error-prone, and time-consuming process. This work fills this significant gap and makes resistance evaluation against differential cryptanalysis easily accessible to cryptographers.

There are other cryptography-specific languages such as SAW [Carter et al. 2013], Jasmin [Almeida et al. 2017], Vale [Bond et al. 2017], Usuba [Mercadier and Dagand 2019], FaCT [Cauligi et al. 2019], QMVerif [Gao et al. 2022], HOME [Gao et al. 2021], and FISCHER [Liu et al. 2023]. They are designed to ensure functional correctness and/or side-channel security of cryptographic algorithms or implementations, which are considerably different from EasyBC.

10 CONCLUSION

We have designed a high-level cryptography-specific language EasyBC for describing block ciphers and presented a rigorous MILP generation procedure from EasyBC programs in the form of differential denotational semantics, leading to a generic and extensible approach for automatically evaluating the resistance of block ciphers written in EasyBC against differential cryptanalysis. We have implemented our approach in an open-source tool and extensively evaluate it on a set of realistic cryptographic primitives, demonstrating its expressivity and capability. In particular, experimental results show that realistic cryptographic primitives can be easily described in EasyBC and their resistance against differential cryptanalysis can be efficiently and effectively proved using our tool. Our tool enables cryptanalysts to easily assess the resistance of block ciphers against differential cryptanalysis in a fully automatic way.

For future research, it would be interesting to improve efficiency by combining recent optimization strategies and to develop analysis approaches for other powerful cryptanalysis (e.g., linear cryptanalysis [Biryukov and De Cannière 2011], impossible differential cryptanalysis [Kim et al. 2003]) based on EasyBC.

ACKNOWLEDGEMENT

We thank the reviewers of POPL'24 for their constructive and insightful comments. This work is supported by the National Natural Science Foundation of China (NSFC) under Grants No. 62072309 and No. 61872340, CAS Project for Young Scientists in Basic Research (YSBR-040), ISCAS New Cultivation Project (ISCAS-PYFX-202201), overseas grants from the State Key Laboratory of Novel Software Technology, Nanjing University (KFKT2023A04), and Birkbeck BEI School Project (EFFECT).

DATA AVAILABILITY STATEMENT

The full version of the paper [Sun et al. 2023] contains missing proofs and more experimental results. Our tool is available at https://github.com/S3L-official/EasyBC.

REFERENCES

Ahmed Abdelkhalek, Yu Sasaki, Yosuke Todo, Mohamed Tolba, and Amr M Youssef. 2017. MILP modeling for (large) s-boxes to optimize probability of differential characteristics. *IACR Transactions on Symmetric Cryptology* (2017), 99–129. https://doi.org/10.13154/TOSC.V2017.I4.99-129

José Bacelar Almeida, Manuel Barbosa, Gilles Barthe, Arthur Blot, Benjamin Grégoire, Vincent Laporte, Tiago Oliveira, Hugo Pacheco, Benedikt Schmidt, and Pierre-Yves Strub. 2017. Jasmin: High-Assurance and High-Speed Cryptography. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1807–1823. https://doi.org/10.1145/3133956.3134078

Kazumaro Aoki, Kunio Kobayashi, and Shiho Moriai. 1997. Best differential characteristic search of FEAL. In *Proceedings of the International Workshop on Fast Software Encryption*. 41–53. https://doi.org/10.1007/BFB0052333

Jean-Philippe Aumasson, Philipp Jovanovic, and Samuel Neves. 2014. Analysis of NORX: Investigating Differential and Rotational Properties. In *Proceedings of the 3rd International Conference on Cryptology and Information Security in Latin America*. 306–324. https://doi.org/10.1007/978-3-319-16295-9_17

Seyyed Arash Azimi, Adrián Ranea, Mahmoud Salmasizadeh, Javad Mohajeri, Mohammad Reza Aref, and Vincent Rijmen. 2022. A bit-vector differential model for the modular addition by a constant and its applications to differential and impossible-differential cryptanalysis. *Des. Codes Cryptogr.* 90, 8 (2022), 1797–1855. https://doi.org/10.1007/S10623-022-01074-8

Subhadeep Banik, Avik Chakraborti, Tetsu Iwata, Kazuhiko Minematsu, Mridul Nandi, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, and Yosuke Todo. 2020. GIFT-COFB. *IACR Cryptol. ePrint Arch.* (2020), 738.

Subhadeep Banik, Sumit Kumar Pandey, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, and Yosuke Todo. 2017. GIFT: A Small Present - Towards Reaching the Limit of Lightweight Encryption. In *Proceedings of the 19th International Conference on Cryptographic Hardware and Embedded Systems*. 321–345. https://doi.org/10.1007/978-3-319-66787-4_16

Zhenzhen Bao, Avik Chakraborti, Nilanjan Datta, Jian Guo, Mridul Nandi, Thomas Peyrin, and Kan Yasuda. 2019. PHOTON-beetle authenticated encryption and hash family. *NIST Lightweight Compet. Round* (2019), 115.

Zhenzhen Bao, Wentao Zhang, and Dongdai Lin. 2014. Speeding up the search algorithm for the best differential and best linear trails. In *Proceedings of the International Conference on Information Security and Cryptology*. 259–285. https://doi.org/10.1007/978-3-319-16745-9_15

Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. 2015. The SIMON and SPECK lightweight block ciphers. In *Proceedings of the 52nd Annual Design Automation Conference*. 175:1–175:6. https://doi.org/10.1145/2744769.2747946

Christof Beierle, Alex Biryukov, Luan Cardoso dos Santos, Johann Großschädl, Léo Perrin, Aleksei Udovenko, Vesselin Velichkov, and Qingju Wang. 2020. Alzette: A 64-Bit ARX-box - (Feat. CRAX and TRAX). In *Proceedings of 40th Annual InternationalCryptology Conference*. 419–448. https://doi.org/10.1007/978-3-030-56877-1_15

Christof Beierle, Alex Biryukov, Luan Cardoso dos Santos, Johann Großschädl, Léo Perrin, Aleksei Udovenko, Vesselin Velichkov, Qingju Wang, and Alex Biryukov. 2019. Schwaemm and Esch: lightweight authenticated encryption and hashing using the sparkle permutation family. *NIST round* 2 (2019).

Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. 2016. The SKINNY family of block ciphers and its low-latency variant MANTIS. In *Proceedings of the Annual International Cryptology Conference*. 123–153. https://doi.org/10.1007/978-3-662-53008-5_5

Tim Beyne, Yu Long Chen, Christoph Dobraunig, and Bart Mennink. 2020. Dumbo, Jumbo, and Delirium: Parallel Authenticated Encryption for the Lightweight Circus. *IACR Trans. Symmetric Cryptol.* (2020), 5–30. https://doi.org/10.13154/TOSC.V2020.IS1.5-30

Eli Biham and Adi Shamir. 1990. Differential Cryptanalysis of DES-like Cryptosystems. In *Proceedings of the 10th Annual International Cryptology Conference*. 2–21. https://doi.org/10.1007/3-540-38424-3_1

Alex Biryukov and Christophe De Cannière. 2011. Linear cryptanalysis for block ciphers. *Encyclopedia of cryptography and security* (2011), 722–725. https://doi.org/10.1007/978-1-4419-5906-5_589

Alex Biryukov and Ivica Nikolić. 2010. Automatic search for related-key differential characteristics in byte-oriented block ciphers: Application to AES, Camellia, Khazad and others. In *Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques*. 322–344. https://doi.org/10.1007/978-3-642-13190-5_17

Nikolaj S. Bjørner and Anh-Dung Phan. 2014. *νZ* - Maximal Satisfaction with Z3. In *Proceedings of the 6th International Symposium on Symbolic Computation in Software Science*. 1–9. https://doi.org/10.29007/JMXJ

Nikolaj S. Bjørner, Anh-Dung Phan, and Lars Fleckenstein. 2015. *νZ* - An Optimizing SMT Solver. In *Proceedings of the 21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. 194–199. https://doi.org/10.1007/978-3-662-46681-0_14

Andrey Bogdanov. 2010. *Analysis and design of block cipher constructions*. Ph. D. Dissertation. Ruhr University Bochum.

Andrey Bogdanov, Lars R Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew JB Robshaw, Yannick Seurin, and Charlotte Vikkelsoe. 2007. PRESENT: An ultra-lightweight block cipher. In *Proceedings of the International workshop on cryptographic hardware and embedded systems*. 450–466. https://doi.org/10.1007/978-3-540-74735-2_31

Barry Bond, Chris Hawblitzel, Manos Kapritsos, K. Rustan M. Leino, Jacob R. Lorch, Bryan Parno, Ashay Rane, Srinath T. V. Setty, and Laure Thompson. 2017. Vale: Verifying High-Performance Cryptographic Assembly Code. In *Proceedings of the 26th USENIX Security Symposium*, Engin Kirda and Thomas Ristenpart (Eds.). 917–934.

Christina Boura and Daniel Coggia. 2020. Efficient MILP modelings for Sboxes and linear layers of SPN ciphers. *IACR Transactions on Symmetric Cryptology* (2020), 327–361. https://doi.org/10.13154/TOSC.V2020.I3.327-361

Kyle Carter, Adam Foltzer, Joe Hendrix, Brian Huffman, and Aaron Tomb. 2013. SAW: the software analysis workbench. In *Proceedings of the 2013 ACM SIGAda annual conference on High integrity language technology*, Jeff Boleng and S. Tucker Taft (Eds.). ACM, 15–18. https://doi.org/10.1145/2527269.2527277

Sunjay Cauligi, Gary Soeller, Brian Johannesmeyer, Fraser Brown, Riad S. Wahby, John Renner, Benjamin Grégoire, Gilles Barthe, Ranjit Jhala, and Deian Stefan. 2019. FaCT: a DSL for timing-sensitive computation. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, Kathryn S. McKinley and Kathleen Fisher (Eds.). ACM, 174–189. https://doi.org/10.1145/3314221.3314605

Zhan Chen, Ning Wang, and Xiaoyun Wang. 2015. Impossible Differential Cryptanalysis of Reduced Round SIMON. *IACR Cryptol. ePrint Arch.* (2015), 286.

Tingting Cui, Keting Jia, Kai Fu, Shiyao Chen, and Meiqin Wang. 2016. New Automatic Search Tool for Impossible Differentials and Zero-Correlation Linear Approximations. *IACR Cryptol. ePrint Arch.* (2016), 689.

Joan Daemen, Seth Hoffert, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. 2020. Xoodyak, a lightweight cryptographic scheme. *IACR Trans. Symmetric Cryptol.* (2020), 60–87. https://doi.org/10.13154/TOSC.V2020.IS1.60-87

Joan Daemen and Vincent Rijmen. 1999. AES proposal: Rijndael. (1999).

Albert Danial. 2021. *cloc: v1.92*. https://doi.org/10.5281/zenodo.5760077

Christoph Dobraunig, Maria Eichlseder, Stefan Mangard, Florian Mendel, Bart Mennink, Robert Primas, and Thomas Unterluggauer. 2020. ISAP v2.0. *IACR Trans. Symmetric Cryptol.* (2020), 390–416. https://doi.org/10.13154/TOSC.V2020.IS1.390-416

Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. 2016. ASCON v1. 2. *Submission to the CAESAR Competition* (2016).

Dirk Fox. 2000. Data Encryption Standard (DES). *Datenschutz und Datensicherheit* 24, 12 (2000).

Kai Fu, Meiqin Wang, Yinghua Guo, Siwei Sun, and Lei Hu. 2016. MILP-based automatic search algorithms for differential and linear trails for speck. In *Proceedings of the International Conference on Fast Software Encryption*. 268–288. https://doi.org/10.1007/978-3-662-52993-5_14

Pengfei Gao, Hongyi Xie, Fu Song, and Taolue Chen. 2021. A Hybrid Approach to Formal Verification of Higher-Order Masked Arithmetic Programs. *ACM Trans. Softw. Eng. Methodol.* 30, 3 (2021), 26:1–26:42. https://doi.org/10.1145/3428015

Pengfei Gao, Hongyi Xie, Pu Sun, Jun Zhang, Fu Song, and Taolue Chen. 2022. Formal Verification of Masking Countermeasures for Arithmetic Programs. *IEEE Trans. Software Eng.* 48, 3 (2022), 973–1000. https://doi.org/10.1109/TSE.2020.3008852

Zheng Gong, Svetla Nikova, and Yee Wei Law. 2011. KLEIN: a new family of lightweight block ciphers. In *Proceedings of the International Workshop on Radio Frequency Identification: Security and Privacy Issues*. 1–18. https://doi.org/10.1007/978-3-642-25286-0_1

LLC Gurobi Optimization. 2018. Gurobi optimizer reference manual.

Martin Hell, Thomas Johansson, Alexander Maximov, Willi Meier, and Hirotaka Yoshida. 2021. Grain-128AEADv2: Strengthening the Initialization Against Key Reconstruction. In *Proceedings of the 20th International Conference on Cryptology and Network Security*. 24–41. https://doi.org/10.1007/978-3-030-92548-2_2

Howard M Heys. 2002. A tutorial on linear and differential cryptanalysis. *Cryptologia* (2002), 189–221. https://doi.org/10.1080/0161-110291890885

Howard M. Heys and Stafford E. Tavares. 1996. Substitution-Permutation Networks Resistant to Differential and Linear Cryptanalysis. *J. Cryptol.* (1996), 1–19. https://doi.org/10.1007/BF02254789

Murat Burhan Ilter and Ali Aydin Selçuk. 2021. A New MILP Model for Matrix Multiplications with Applications to KLEIN and PRINCE. In *Proceedings of the 18th International Conference on Security and Cryptography*. 420–427. https://doi.org/10.5220/0010519504200427

Tetsu Iwata, Mustafa Khairallah, Kazuhiko Minematsu, and Thomas Peyrin. 2020. Duel of the Titans: The Romulus and Remus Families of Lightweight AEAD Algorithms. *IACR Trans. Symmetric Cryptol.* (2020), 43–120. https://doi.org/10.13154/TOSC.V2020.I1.43-120

Maryam Izadi, Babak Sadeghiyan, Seyed Saeed Sadeghian, and Hossein Arabnezhad Khanooki. 2009. MIBS: A New Lightweight Block Cipher. In *Proceedings of the 8th International Conference on Cryptology and Network Security*. 334–348. https://doi.org/10.1007/978-3-642-10433-6_22

Fulei Ji, Wentao Zhang, and Tianyou Ding. 2021. Improving matsui's search algorithm for the best differential/linear trails and its applications for DES, DESL and GIFT. *Comput. J.* 64, 4 (2021), 610–627. https://doi.org/10.1093/COMJNL/BXAA090

John B. Kam and George I. Davida. 1979. Structured design of substitution-permutation encryption networks. *IEEE Trans. Comput.* 28, 10 (1979), 747–753. https://doi.org/10.1109/TC.1979.1675242

Jonathan Katz and Yehuda Lindell. 2014. *Introduction to Modern Cryptography, 2nd Edition.* CRC Press.

Jongsung Kim, Seokhie Hong, Jaechul Sung, Sangjin Lee, Jongin Lim, and Soohak Sung. 2003. Impossible differential cryptanalysis for block cipher structures. In *Proceedings of the International Conference on Cryptology in India*. 82–96. https://doi.org/10.1007/978-3-540-24582-7_6

Lars R. Knudsen. 1997. Block Ciphers - A Survey. In *Proceedings of the State of the Art in Applied Cryptography, Course on Computer Security and Industrial Cryptography*, Bart Preneel and Vincent Rijmen (Eds.), Vol. 1528. Springer, 18–48. https://doi.org/10.1007/3-540-49248-8_2

Stefan Kölbl, Gregor Leander, and Tyge Tiessen. 2015. Observations on the SIMON Block Cipher Family. In *Proceedings of the 35th Annual Cryptology Conference*. Springer, 161–185. https://doi.org/10.1007/978-3-662-47989-6_8

Xuejia Lai, James L Massey, and Sean Murphy. 1991. Markov ciphers and differential cryptanalysis. In *Proceedings of the Workshop on the Theory and Application of of Cryptographic Techniques*. Springer, 17–38. https://doi.org/10.1007/3-540-46416-6_2

Lingchen Li, Wenling Wu, Yafei Zheng, and Lei Zhang. 2019. The Relationship between the Construction and Solution of the MILP Models and Applications. *IACR Cryptol. ePrint Arch.* (2019), 49.

Ting Li and Yao Sun. 2022. SuperBall: A New Approach for MILP Modelings of Boolean Functions. *IACR Transactions on Symmetric Cryptology* 2022, 3 (2022), 341–367. https://doi.org/10.46586/TOSC.V2022.I3.341-367

Mingyang Liu, Fu Song, and Taolue Chen. 2023. Automated Verification of Correctness for Masked Arithmetic Programs. In *Proceedings of the 35th International Conference on Computer Aided Verification (CAV), Part III (Lecture Notes in Computer Science, Vol. 13966)*, Constantin Enea and Akash Lal (Eds.). Springer, 255–280. https://doi.org/10.1007/978-3-031-37709-9_13

Yu Liu, Huicong Liang, Muzhou Li, Luning Huang, Kai Hu, Chenhe Yang, and Meiqin Wang. 2021. STP models of optimal differential and linear trail for S-box based ciphers. *Science China Information Sciences* 64, 5 (2021). https://doi.org/10.1007/S11432-018-9772-0

Zhengbin Liu, Yongqiang Li, and Mingsheng Wang. 2017. Optimal Differential Trails in SIMON-like Ciphers. *IACR Trans. Symmetric Cryptol.* 2017 (2017), 358–379. https://doi.org/10.13154/TOSC.V2017.I1.358-379

Mohammad Mahzoun, Liliya Kraleva, Raluca Posteuca, and Tomer Ashur. 2022. Differential Cryptanalysis of K-Cipher. In *IEEE Symposium on Computers and Communications*. 1–7. https://doi.org/10.1109/ISCC55528.2022.9912926

Rusydi H Makarim and Raghvendra Rohit. 2022. Towards Tight Differential Bounds of ASCON: A Hybrid Usage of SMT and MILP. *IACR Transactions on Symmetric Cryptology* (2022), 303–340. https://doi.org/10.46586/TOSC.V2022.I3.303-340

Mitsuru Matsui. 1994. On Correlation Between the Order of S-boxes and the Strength of DES. In *Advances in Cryptology - EUROCRYPT '94, Workshop on the Theory and Application of Cryptographic Techniques, Perugia, Italy, May 9-12, 1994, Proceedings (Lecture Notes in Computer Science, Vol. 950)*, Alfredo De Santis (Ed.). Springer, 366–375. https://doi.org/10.1007/BFB0053451

Darius Mercadier and Pierre-Évariste Dagand. 2019. Usuba: high-throughput and constant-time ciphers, by construction. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, Kathryn S. McKinley and Kathleen Fisher (Eds.). ACM, 157–173. https://doi.org/10.1145/3314221.3314636

Nicky Mouha and Bart Preneel. 2013. Towards finding optimal differential characteristics for ARX: Application to Salsa20. *Cryptology ePrint Archive* (2013).

Nicky Mouha, Qingju Wang, Dawu Gu, and Bart Preneel. 2011. Differential and linear cryptanalysis using mixed-integer linear programming. In *Proceedings of the International Conference on Information Security and Cryptology*. 57–76.

https://doi.org/10.1007/978-3-642-34704-7_5

Vu Nguyen, Sophia Deeds-Rubin, Thomas Tan, and Barry Boehm. 2007. A SLOC counting standard. In *Cocomo ii forum*, Vol. 2007. 1–16.

NIST. 2023. Finalists of NIST lightweight cryptography standardization process. https://csrc.nist.gov/Projects/lightweight-cryptography/finalists.

Kaisa Nyberg. 1996. Generalized feistel networks. In *Proceedings of the International conference on the theory and application of cryptology and information security*. 91–104.

Yu Sasaki and Yosuke Todo. 2017. New algorithm for modeling S-box in MILP based differential and division trail search. In *Proceedings of the International Conference for Information Technology and Communications*. 150–165. https://doi.org/10.1007/978-3-319-69284-5_11

Claude E. Shannon. 1949. Communication theory of secrecy systems. *Bell System Technical Journal* 28, 4 (1949), 656–715.

Kyoji Shibutani, Takanori Isobe, Harunaga Hiwatari, Atsushi Mitsuda, Toru Akishita, and Taizo Shirai. 2011. Piccolo: an ultra-lightweight blockcipher. In *Proceedings of the International workshop on cryptographic hardware and embedded systems*. 342–357. https://doi.org/10.1007/978-3-642-23951-9_23

Ling Song, Zhangjie Huang, and Qianqian Yang. 2016. Automatic Differential Analysis of ARX Block Ciphers with Application to SPECK and LEA. In *Proceedings of the 21st Australasian Conference on Information Security and Privacy*. 379–394. https://doi.org/10.1007/978-3-319-40367-0_24

Ling Sun, Wei Wang, and Meiqin Wang. 2018. More Accurate Differential Properties of LED64 and Midori64. *IACR Trans. Symmetric Cryptol.* 2018, 3 (2018), 93–123. https://doi.org/10.13154/TOSC.V2018.I3.93-123

Ling Sun, Wei Wang, and Meiqin Wang. 2021. Accelerating the Search of Differential and Linear Characteristics with the SAT Method. *IACR Trans. Symmetric Cryptol.* 2021, 1 (2021), 269–315. https://doi.org/10.46586/TOSC.V2021.I1.269-315

Pu Sun, Fu Song, Yuqi Chen, and Taolue Chen. 2023. *EasyBC: A Cryptography-Specific Language for Security Analysis of Block Ciphers against Differential Cryptanalysis (Full version).* Technical Report. https://github.com/S3L-official/EasyBC.

Siwei Sun, Lei Hu, Ling Song, Yonghong Xie, and Peng Wang. 2013. Automatic security evaluation of block ciphers with S-bP structures against related-key differential attacks. In *Proceedings of the International Conference on Information Security and Cryptology*. 39–51. https://doi.org/10.1007/978-3-319-12087-4_3

Siwei Sun, Lei Hu, Meiqin Wang, Peng Wang, Kexin Qiao, Xiaoshuang Ma, Danping Shi, Ling Song, and Kai Fu. 2014b. Towards finding the best characteristics of some bit-oriented block ciphers and automatic enumeration of (related-key) differential and linear characteristics with predefined properties. *Cryptology ePrint Archive* (2014).

Siwei Sun, Lei Hu, Peng Wang, Kexin Qiao, Xiaoshuang Ma, and Ling Song. 2014a. Automatic security evaluation and (related-key) differential characteristic search: application to SIMON, PRESENT, LBlock, DES(L) and other bit-oriented block ciphers. In *Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security*. 158–178. https://doi.org/10.1007/978-3-662-45611-8_9

Yao Sun. 2021. Towards the Least Inequalities for Describing a Subset in Z_2n. *IACR Cryptol. ePrint Arch.* (2021), 1084.

Tomoyasu Suzaki, Kazuhiko Minematsu, Sumio Morioka, and Eita Kobayashi. 2012. TWINE: A Lightweight Block Cipher for Multiple Platforms. In *Proceedings of the International Conference on Selected Areas in Cryptography*. 339–354. https://doi.org/10.1007/978-3-642-35999-6_22

Je Sen Teh and Alex Biryukov. 2022. Differential cryptanalysis of WARP. *J. Inf. Secur. Appl.* 70 (2022), 103316. https://doi.org/10.1016/J.JISA.2022.103316

Aleksei Udovenko. 2021. MILP modeling of Boolean functions by minimum number of inequalities. *Cryptology ePrint Archive* (2021).

Xuzi Wang, Baofeng Wu, Lin Hou, and Dongdai Lin. 2018. Automatic Search for Related-Key Differential Trails in SIMON-like Block Ciphers Based on MILP. In *Proceedings of the 21st International Conference on Information Security*, Liqun Chen, Mark Manulis, and Steve A. Schneider (Eds.). 116–131. https://doi.org/10.1007/978-3-319-99136-8_7

Hongjun Wu and Tao Huang. 2019. TinyJAMBU: A family of lightweight authenticated encryption algorithms. *Submission to the NIST Lightweight Cryptography Standardization Process* (2019).

Shengbao Wu and Mingsheng Wang. 2012. Automatic search of truncated impossible differentials for word-oriented block ciphers. In *Proceedings of the International Conference on Cryptology in India*. 283–302.

Wenling Wu and Lei Zhang. 2011. LBlock: a lightweight block cipher. In *Proceedings of the International conference on applied cryptography and network security*. 327–344. https://doi.org/10.1007/978-3-642-21554-4_19

Jun Yin, Chuyan Ma, Lijun Lyu, Jian Song, Guang Zeng, Chuangui Ma, and Fushan Wei. 2017. Improved cryptanalysis of an ISO standard lightweight block cipher with refined MILP modelling. In *Proceedings of the International Conference on Information Security and Cryptology*. 404–426. https://doi.org/10.1007/978-3-319-75160-3_24

Pei Zhang and Wenying Zhang. 2018. Differential cryptanalysis on block cipher skinny with MILP program. *Security and Communication Networks* 2018 (2018). https://doi.org/10.1155/2018/3780407

Wentao Zhang, Zhenzhen Bao, Dongdai Lin, Vincent Rijmen, Bohan Yang, and Ingrid Verbauwhede. 2015. RECTANGLE: a bit-slice lightweight block cipher suitable for multiple platforms. *Science China Information Sciences* 58, 12 (2015), 1–15.

https://doi.org/10.1007/S11432-015-5459-7

Yingjie Zhang, Siwei Sun, Jiahao Cai, and Lei Hu. 2018. Speeding up MILP aided differential characteristic search with Matsui's strategy. In *Proceedings of the International Conference on Information Security*. 101–115. https://doi.org/10.1007/978-3-319-99136-8_6

Chunning Zhou, Wentao Zhang, Tianyou Ding, and Zejun Xiang. 2019. Improving the MILP-based security evaluation algorithm against differential/linear cryptanalysis using a divide-and-conquer approach. *IACR Transactions on Symmetric Cryptology* (2019), 438–469. https://doi.org/10.13154/TOSC.V2019.I4.438-469

Received 2023-07-11; accepted 2023-11-07