Characteristic Automated Search of Cryptographic Algorithms for Distinguishing Attacks (CASCADA)

Adrián Ranea[1] and Vincent Rijmen[1]

imec-COSIC, KU Leuven, Belgium
firstname.lastname@esat.kuleuven.be

Abstract Automated search methods based on Satisfiability Modulo Theories (SMT) problems are being widely used to evaluate the security of block ciphers against distinguishing attacks. While these methods provide a systematic and generic methodology, most of their software implementations are limited to a small set of ciphers and attacks, and extending these implementations requires significant effort and expertise. In this work we present `CASCADA`, an open-source Python library to evaluate the security of cryptographic primitives, specially block ciphers, against distinguishing attacks with bit-vector SMT solvers. The tool `CASCADA` implements the bit-vector property framework herein proposed and several SMT-based automated search methods to evaluate the security of ciphers against differential, related-key differential, rotational-XOR, impossible-differential, impossible-rotational-XOR, related-key impossible-differential, linear and zero-correlation cryptanalysis. The library `CASCADA` is the result of a huge engineering effort, and it provides many functionalities, a modular design, an extensive documentation and a complete suite of tests.

Keywords: cryptanalysis · automated search · SMT · bit-vector theory

1 Introduction

Automated tools have gained significant traction in the last decade in the security evaluation of cryptographic algorithms, specially block ciphers. In the design and cryptanalysis of a block cipher, the security is evaluated by verifying that no known attacks cannot efficiently recover the secret key of the cipher. Most cipher attacks include an initial distinguishing attack, where a non-random property of the cipher is exploited to distinguish the cipher from a random permutation. The distinguishing step is followed by a key-recovery step, but finding the exploitable property of the cipher is the hardest part to mount the attack.

Well-known examples of powerful cipher attacks including a distinguishing step are differential [BS93], related-key differential [Bih94], impossible-differential [BBS99a] and rotational-XOR (RX) [AL16] cryptanalysis, and also

linear [Mat93] and zero-correlation cryptanalysis [BR14]. The properties exploited by (related-key) differential and (related-key) impossible-differential cryptanalysts are (related-key) differential objects, by RX cryptanalysis are RX difference pairs, and by linear and zero-correlation cryptanalysis are linear approximations. In the next section we will introduce these properties in more detail.

While these properties were traditionally searched for ad-hoc and manually, recent works proposed the use of automated tools based on constraint satisfaction problems, such as SMT (Satisfiability Modulo Theories) or MILP (Mixed Integer Linear Programming) [MP13; Mou+11]. Automated methods model these searches as constraint satisfaction problems and solve them with powerful off-the-shelf solvers available nowadays [ÁK16; BT18; Lod10], freeing designers and cryptanalysts from the effort of implementing and optimizing the search.

Unfortunately, most automated methods published in the literature do not provide software implementations [SHY16; LWR16; Abd+17; ST17; Yin+17; Wan+18; Roh+19] or provide narrow implementations that are specific to a cipher and a distinguishing attack [Mou+11; MP13; Sun+14; Lu+20], and extending these implementations to other ciphers or attacks require significant effort and expertise.

The notable exceptions are the SMT-based tools `CryptoSMT` [Ste] and `ArxPy` [Ran]. Both libraries support many block ciphers and several distinguishing attacks, namely differential and linear cryptanalysis in `CryptoSMT` and (related-key) differential, RX, and (related-key) impossible-differential in `ArxPy`, and they have been used in multiple works, e.g., [KLT15; AK18; AL18; Had+19] or [RLA17; KPR20; Azi+20; Azi+22]. However, these two libraries present severe limitations. For example, `CryptoSMT` lacks code documentation and tests, and adding a new cipher in `CryptoSMT` requires significant expertise and effort as one needs to implement the differential and linear SMT models of the cipher. The tool `ArxPy` does not suffer from these limitations, but only supports Addition-Rotation-XOR (ARX) ciphers and does not support linear or zero-correlation cryptanalysis.

Contributions. In this work we present `CASCADA` [Ran22] (Characteristic Automated Search of Cryptographic Algorithms for Distinguishing Attacks), an open-source Python library to evaluate the security of block ciphers and other cryptographic primitives against several distinguishing attacks by searching for exploitable properties using bit-vector SMT solvers. `CASCADA` is available at `https://github.com/ranea/CASCADA`.

The library `CASCADA` is based on `ArxPy`, but only a third of `CASCADA`'s source code derives from `ArxPy` and `CASCADA` implements more automated methods and distinguishing attacks, supports a wider class of block ciphers and primitives, and improves the code, documentation and tests.

The tool `CASCADA` implements the search for differentials, RX difference pairs and linear approximations to be used in differential, related-key differential, impossible-differential, related-key impossible-differential, RX, impossible-RX, linear, and zero-correlation cryptanalysis. The automated search for these properties is implemented in `CASCADA` following the bit-vector property framework herein proposed, so that other distinguishing attacks can be easily added.

Four automated search methods based on bit-vector SMT problems are implemented in `CASCADA`. Three of these methods are the generalization of previous work [MP13; ST17; Azi+22] to the bit-vector property framework, and the fourth method is a new automated search method based on quantified SMT problems.

The open-source library `CASCADA` is the result of a huge engineering effort aiming to provide a state-of-the-art tool to evaluate a wide class of cryptographic algorithms against many distinguishing attacks. To this end, `CASCADA` features a modular design, an extensive documentation and a complete suite of tests so that `CASCADA` is not only easy to use but also to extend by designers and cryptanalysts.

Outline. In Sect. 2 the preliminaries are introduced, and in Sect. 3 the bit-vector property framework is presented. Automated methods based on bit-vector SMT problems are explained in Sect. 4, and Sect. 5 describes the functionality and implementation of `CASCADA`.

2 Preliminaries

2.1 Bit-vector SMT Problems

A bit-vector expression is a bit-vector constant, a bit-vector variable or a bit-vector operation with bit-vector expressions as inputs. Bit-vector constants are interpreted as unsigned integers in base 2; the n-bit vector $x = b_{n-1} \cdots b_1 b_0$ denotes the non-negative integer $b_0 + 2b_1 + \cdots + 2^{n-1}b_{n-1}$. The i-th bit of x, b_i, is also denoted by $x[i]$, where $x[0] = b_0$ denotes the Least Significant Bit (LSB) and $x[n-1] = b_{n-1}$ denotes the Most Significant Bit (MSB). We consider here the following bit-vector operations and notations:

- The concatenation and extraction of bit-vectors.
- The bit-wise logical operations: negation $\neg$, conjunction $\wedge$, disjunction $\vee$, and exclusive-or (XOR) $\oplus$.
- The shift operations: left shift $\ll$, (logical) right shift $\gg$, circular left rotation $\lll$ and circular right rotation $\ggg$.
- The arithmetical operations: modular addition $\boxplus$, modular subtraction $\boxminus$, modular multiplication, $\boxtimes$ unsigned truncated division operation $\boxdiv$ and unsigned remainder (modulus) operation $\boxpercent$.
- The relational operations: $=, <, >, \leq$ and $\geq$.
- The if-then-else operator $\mathrm{Ite}(b, x, y)$, returning x if b is equal to the bit `0` and otherwise returning y.

A bit-vector formula or constraint is a bit-vector expression returning a single bit, where the bit `0` denotes the truth value `False` and the bit `1` denotes `True`. A bit-vector formula is satisfiable if there is an assignment of the variables that makes the formula `True`.

Satisfiability Modulo Theories (SMT) refers to the problem of determining whether a first order formula is satisfiable with respect to some logical theory

[ÁK16; BT18]. SMT problems can be seen as a generalization of SAT problems; the latter problems are expressed in propositional logic, and SMT problems are given in richer logics such as the theory of integers or the theory of bit-vectors.

An SMT problem defined in the bit-vector theory, or simply a bit-vector SMT problem, is given by a list of bit-vector variables, each one associated with the existential $\exists$ or for-all $\forall$ quantifier, and a list of bit-vector constraints including these variables. An SMT problem where the quantifiers are not specified is called a quantifier-free problem, and it is equisatisfiable to the same SMT problem with existential quantifiers. On the other hand, SMT problems combining existential and for-all quantifiers are called quantified problems, and these are much harder to solve [Rey16].

For example, given the function $f_k(x,y) = \big(((x \ggg 7) \boxplus y) \oplus k, (y \lll 2) \oplus (((x \ggg 7) \boxplus y) \oplus k)\big)$ with $x,y,k \in \{0,1\}^{16}$, the decision problem of determining whether there exists an assignment of k such that $f_k(0,0) = (0,0)$ can be written as the following bit-vector SMT problem

$$\begin{aligned} \exists x, y, k, x', y' \in \{0,1\}^{16} : \\ x' &= ((x \ggg 7) \boxplus y) \oplus k \\ y' &= (y \lll 2) \oplus x' \\ 0 &= x \vee y \vee x' \vee y' \,. \end{aligned}$$

Software tools that determine the satisfiability of SMT problems are called SMT solvers. In the past two decades, SMT solvers have grown in popularity due to technological advances and industrial applications in software engineering, optimization, and many other areas [MB11]. On top of that, many state-of-the-art and open-source SMT solvers are available nowadays such as Boolector [NPB14] or STP [GD07], among others.

SMT solvers not only can determine the satisfiability of an SMT problem but also find an assignment of the variables that satisfies the problem. This feature allows SMT solvers to be used in search problems. Following our previous bit-vector SMT example, by using an SMT solver supporting the bit-vector theory, we could first check whether the problem is satisfiable, and in that case, find a value of k that makes $f_k(0,0) = (0,0)$.

Most SMT solvers supporting the bit-vector theory support the bit-vector core theory of the SMT-LIB Standard 2.0 [BST10]. This standard includes most of the bit-vector operations herein considered, and the ones that are not included (e.g., Ite) can be easily defined from the operations in the standard. Thus, we consider here bit-vector SMT problems built from our list of bit-vector operations, and these SMT problems can be given to any SMT solver supporting SMT-LIB Standard 2.0, such as Boolector or STP.

2.2 Distinguishing Attacks on Block Ciphers

A block cipher is a family of permutations $\{E_k\}_k$ parametrized by a key $k \in \mathcal{K}$, where E_k maps n-bit plaintexts to n-bit ciphertexts, and both E_k and E_k^{-1} can

be efficiently computed. In an iterated block cipher, the encryption function E_k is built as the composition of round functions, i.e., $E_k = f_{r-1} \circ f_{r-2} \circ \cdots \circ f_0$, where a list of round keys are derived from a key-scheduling algorithm $\mathrm{KS}(k) = (k_0, k_1, \ldots, k_{r-1})$ and the i-th round key k_i is injected in the i-th round function f_i.

Informally, the security of a block cipher is argued by showing that known attacks cannot efficiently recover the key. Most of the powerful attacks against block ciphers contain a distinguishing attack, where a non-random property of the cipher is exploited to distinguish the cipher from a random permutation. The distinguishing attack is usually followed by a key-recovery attack, but finding the exploitable property for the distinguishing attack is the crucial part and the focus of this work.

The properties exploited in differential cryptanalysis are differentials (α, β) over the encryption function E_k with high expected differential probability. Given a differential (α, β) over f, its differential probability is given by

$$\#\{x : f(x \vartriangle \alpha) \triangledown f(x) = \beta\}/2^n\,, \tag{1}$$

where usually $\vartriangle = \oplus = \triangledown$. The $\triangledown$ operator computes the difference of a pair of values (x, x'), and the $\vartriangle$ operator takes as input a value x and a difference α and outputs the value x' such that the pair (x, x') have difference α.

The expected differential probability p is the differential probability averaged over the key space $\mathcal{K}$,

$$p = \frac{1}{|\mathcal{K}|} \sum_{k \in \mathcal{K}} \#\{x : f(x \vartriangle \alpha) \triangledown f(x) = \beta\}/2^n\,,$$

and the complexity of differential cryptanalysis is $\mathcal{O}(1/p)$ [BS93]. Related-key differential cryptanalysis is a variant of differential cryptanalysis that exploits related-key differentials (α, κ, β) with high expected related-key differential probability, where the related-key differential probability is given by

$$\#\{x : E_{k \vartriangle \kappa}(x \vartriangle \alpha) \triangledown E_k(x) = \beta\}/2^n\,. \tag{2}$$

We refer the reader to [DR07] for a formal introduction to the notions of differential and differential probability.

Impossible-differential cryptanalysis exploits differentials with zero differential probability for all keys, and similarly related-key impossible-differential cryptanalysis exploits related-key differentials with zero probability. While the complexity of (related-key) impossible-differential cryptanalysis is roughly the cardinality of the input space, this can be significantly reduced by using multiple zero-probability (related-key) differentials [Bou+18].

In RX cryptanalysis, the properties exploited are RX difference pairs $(\Delta_\gamma, \Delta'_\gamma)$ with high expected RX probability, where the RX probability over an n-bit function f is defined as [Lu+20]

$$\sum \#\{x : (f(x) \lll \gamma) \oplus f((x \lll \gamma) \oplus \Delta_\gamma) = \Delta'_\gamma\}/2^n\,.$$

Since an RX difference pair (α, β) is equivalent to a differential (α, β) with $\{\nabla, \triangle\}$ defined as

$$x' \nabla x = x' \oplus (x \lll 1), \quad x \triangle \alpha = (x \lll 1) \oplus \alpha\,, \tag{3}$$

in this paper we will call RX difference pairs RX differentials, and differentials with $\triangle = \oplus = \nabla$ will be called XOR differentials.

On the other hand, linear cryptanalysis exploit linear approximations (α, β) over the encryption function E_k with high expected linear probability or potential. Let $C_f(\alpha, \beta)$ be the correlation of (α, β) over an n-bit function f defined as

$$C_f(\alpha, \beta) = 2 \times (\#\{x : \langle \alpha, x \rangle = \langle \beta, f(x) \rangle\} \;/\; 2^n) - 1\,, \tag{4}$$

where $\langle \cdot, \cdot \rangle$ denotes the inner product. The potential p is the linear probability (square of the correlation) averaged over the key space $\mathcal{K}$,

$$p = \frac{1}{|\mathcal{K}|} \sum_{k \in \mathcal{K}} C_{E_k}(\alpha, \beta)^2\,,$$

and the complexity of linear cryptanalysis is $\mathcal{O}(1/p)$ [Mat93].[1] We refer the reader to [DR07; ABR20] for a formal introduction to the notions of linear approximations and correlations.

Lastly, zero-correlation cryptanalysis exploit linear approximations with zero correlation for all keys. Similar to impossible-differential cryptanalysis, the complexity of zero-correlation cryptanalysis is roughly the cardinality of the plaintext space, and it can be reduced by using multiple zero-correlation linear approximations [BW12].

3 Bit-vector Property Framework

To model systematically the search for (related-key) XOR differentials, RX differentials and linear approximations with bit-vector SMT problems, we will introduce the bit-vector property framework containing the notions of bit-vector property, bit-vector characteristic, bit-vector property model and bit-vector characteristic model. Other frameworks unifying block cipher cryptanalysis have also been proposed [Wag04; PS06], but our simple framework (implemented in `CASCADA`) easily models the search for exploitable properties as bit-vector automated methods.

3.1 Bit-vector Properties and Characteristics

A (bit-vector) property over a function f is a pair of bit-vectors (α, β) with an associated propagation probability $\mathrm{PP}_f(\alpha, \beta) \in [0, 1] \subseteq \mathbb{R}$. In this case, we also say that the input property α propagates to the output property β with probability $\mathrm{PP}_f(\alpha, \beta)$.

[1] Follow-up studies of [Mat93] (e.g., [BT13; ABR20]) provide a more accurate estimation of the complexity of linear cryptanalysis.

For a function f_k depending on external values $k = (k_1, k_2, \dots)$, non-input but unknown fixed values such as round keys, a bit-vector property over f_k can include an additional bit-vector value κ so that the propagation probability depends not only on the input and output properties (α, β) but also on the external property κ.

We consider here three types of properties[2]: the XOR difference property, the RX difference property and the linear mask property. A difference property (α, β) over a function f is defined as the bit-vector property (α, β) over f where the propagation probability is given by Eqn. (1), or by Eqn. (2) and averaged over $\mathcal{K}$ if f contains external values $k \in \mathcal{K}$. XOR difference properties consider $\triangle = \oplus = \triangledown$ and RX difference properties consider $\triangle$ and $\triangledown$ given by Eqn. (3).

A linear mask property (α, β) over f is a bit-vector property (α, β) over f where the propagation probability is given by the absolute value of the correlation $C_f(\alpha, \beta)$ given by Eqn. (4) (averaged over $\mathcal{K}$ if f contains external values $k \in \mathcal{K}$).

While distinguishing attacks only require the global property (α, β) and its propagation probability, computing the propagation probability is a hard problem for complex functions such as block ciphers. The main approach of distinguishing attacks is to analyse the local propagation probabilities of the round functions, to obtain a trail of local properties and to estimate the global propagation probability as the product of the local propagation probabilities.

A (bit-vector) characteristic Γ over $f = f_{r-1} \circ f_{r-2} \circ \cdots \circ f_0$ is a trail of properties $(\gamma_0, \gamma_1, \dots, \gamma_r)$ where (γ_i, γ_{i+1}) is a bit-vector property over f_i. The bit-vectors (γ_0, γ_r) are also called the input and output property respectively of Γ. The propagation probability of Γ is defined as

$$\mathrm{PP}_f(\Gamma) = \mathrm{PP}_{f_0}(\gamma_0, \gamma_1) \times \mathrm{PP}_{f_1}(\gamma_1, \gamma_2) \times \cdots \times \mathrm{PP}_{f_{r-1}}(\gamma_{r-1}, \gamma_r)\,.$$

A characteristic with XOR (resp. RX) properties is called an XOR (RX) differential characteristic, and a characteristic with linear mask properties is called a linear characteristic. A related-key differential characteristic over a block cipher is a pair of differential characteristics $(\Gamma_{\mathrm{KS}}, \Gamma_{E_k})$ with Γ_{KS} defined over the key-schedule function KS and Γ_{E_k} over the encryption function E_k such that the (external) round key properties of Γ_{E_k} are set to the properties of Γ_{KS}.

Depending on the function and the property, the propagation probability of a characteristic $\Gamma = (\gamma_0, \gamma_1, \dots, \gamma_r)$ might not accurately approximate the propagation probability of the global property (γ_0, γ_r), see for example [AK18]. Nevertheless, this approximation is widely used in the design and cryptanalysis of block ciphers (particularly ARX ciphers) due to the lack of other systematic approaches.

In practice block ciphers are claimed secure against distinguishing attacks by showing that no high-probability characteristics and zero-probability global

[2] The tool `CASCADA` also implements a fourth property type, the value property, where the propagation probability of the value property (α, β) over f is 1 if $\beta = f(\alpha)$ and 0 otherwise. The value property is not exploited by the distinguishing attacks herein considered; it is implemented by `CASCADA` to mount straightforward SMT-based key-recovery attacks.

properties can be found, and most of the successful attacks against block ciphers have exploited these objects as well. Thus, systematic methods searching for these objects are crucial for the design and analysis of block ciphers.

To search for high-probability characteristics and zero-probability global properties using bit-vector SMT problems, the propagation probabilities of characteristics and properties need to be encoded as bit-vector constraints. To this end, we will introduce the notions of bit-vector property model and bit-vector characteristic model.

3.2 Bit-vector Property Model

We say a property (α, β) over a function f is valid if its propagation probability is non-zero. In this case, we define the propagation weight of (α, β) as the negative binary logarithm of its propagation probability, that is,

$$\mathrm{PW}_f(\alpha, \beta) = -\log_2(\mathrm{PP}_f(\alpha, \beta)) .$$

A (bit-vector) property model of f is a set of bit-vector constraints that models the propagation weight of properties over f. A property model of f is given by three bit-vector constraints: the validity constraint, the probability-one constraint and the weight constraint.

- The validity constraint with inputs (α, β) is `True` if and only if the property (α, β) is valid.
- The probability-one constraint with inputs (α, β) is `True` if and only if the propagation probability of the property (α, β) is 1.
- The weight constraint with inputs (w, α, β) is `True` if and only if the bit-vector w is equal to the propagation weight of the property (α, β).

The weight constraint is only defined for inputs (α, β) with non-zero propagation probability; the truth value of the weight constraint for invalid (α, β) does not matter. While the probability-one constraint is equivalent to the logical AND of the validity constraint and the weight constraint with input $w = 0$, for many functions it is possible to specifically model the probability-one constraint with a simpler formula rather than with the combination of the validity and weight constraint.

In bit-vector SMT problems, the multiplication $\boxtimes$ is more expensive than the addition $\boxplus$. To avoid modelling the propagation probability of $g \circ f$ as the multiplication of the local probabilities of f and g, a property model includes the weight constraint rather than a probability constraint (a constraint with inputs (p, α, β) being True if p is equal to the propagation probability of (α, β))). Thus, the propagation weight of $g \circ f$ can be efficiently modelled as the sum of the local propagation weights of f and g.

By default, the n_w-bit input w of the weight constraint is interpreted as the non-negative integer $w[0] + 2w[1] + \cdots + 2^{n_w-1}w[n_w - 1]$. However, since the propagation weight can be a non-integer value for some properties and functions, we consider weight constraints where the input w is interpreted as the rational

value $2^{-\ell}(w[0] + 2w[1] + \cdots + 2^{n_w-1}w[n_w - 1])$ for a given fixed number ℓ of fractional bits. Moreover, we also consider weight constraints that are `True` if and only if $|w - \mathrm{PW}_f(\alpha, \beta)| < \epsilon$ for a fixed error bound ϵ.

A property model with respect to the XOR difference, RX difference or linear mask property is called an XOR differential, RX differential or linear model, respectively. To the best of our knowledge, the models for these properties published this far are the following:

- *XOR differential models.* Given a $\oplus$-linear bit-vector function f, an XOR differential model of f is given by the validity and probability-one constraint $\beta = f(\alpha)$ and the weight constraint $w = 0$. XOR differential models of the modular addition $f(x, x') = x \boxplus x'$ were implicitly obtained in [LM01; Sch13], and an XOR differential model of the modular addition with a constant $\boxplus_c(x) = x \boxplus c$ was proposed in [Azi+20]. For the round function of the block cipher Simon [Bea+15], $f_{a,b,c}(x) = (x \lll a) \wedge (x \lll b) \wedge (x \lll c)$, an XOR differential model was obtained in [KLT15].
- *RX differential models.* Given a $\oplus$-linear bit-vector function f that commutes with $\lll_1$, an RX differential model of f is given by the validity and probability-one constraint $\beta = f(\alpha)$ and the weight constraint $w = 0$. An RX differential model of the modular addition was proposed in [AL16], and an RX differential of the Simon round function in [Lu+20].
- *Linear models.* For a $\oplus$-linear function f given by the binary matrix M, a linear model of f is given by the validity and probability-one constraint $\alpha = M^t(\beta)$ and the weight constraint $w = 0$. Linear models of the modular addition were implicitly obtained in [Sch13; LWR16], and a linear model of the Simon round function was proposed in [KLT15].

For a bit-vector function f with small input and output bitsize, one can store the propagation weights of all properties (α, β) in a table and derive the property model of f from this table. This approach has been originally used for XOR differential models [Sun+14; Abd+17; SWW18; AK18], and it can easily be generalized for any bit-vector property.

Given a function with no efficient property model, one can also model with simple and efficient constraints a simplified variant of its propagation probability, where the truth value of the simple constraints is not accurate for some inputs (α, β). We consider here two types of simplified models: weak and branch-based models.

A weak model simplifies the propagation probability by considering only four possible propagation probabilities depending on whether α or β are zero or non-zero. A branch-based model is similar to a weak model but with the additional rule that a non-zero property (α, β) is considered invalid if the number of non-zero words in α and β is strictly lower than a given fixed number B. Usually, B is chosen as the branch number of f, that is, the minimum number of active words among all non-zero properties over f. These simplified models were originally used in [Mou+11] for the XOR difference and the linear mask properties, where weak models were used for the S-boxes and branch-based models were used for the linear layers.

3.3 Bit-vector Characteristic Model

We say a characteristic $\Gamma = (\gamma_0, \gamma_1, \ldots, \gamma_r)$ of a function $f = f_{r-1} \circ f_{r-2} \circ \cdots \circ f_0$ is valid if the propagation probability of Γ is non-zero, and in this case we define the propagation weight of Γ as

$$\mathrm{PW}_f(\Gamma) = -\log_2(\mathrm{PP}_f(\Gamma)) = \mathrm{PW}_{f_0}(\gamma_0, \gamma_1) + \cdots + \mathrm{PW}_{f_{r-1}}(\gamma_{r-1}, \gamma_r)\,.$$

A (bit-vector) characteristic model of $f = f_{r-1} \circ f_{r-2} \circ \cdots \circ f_0$ is a set of bit-vector constraints that models the propagation weight of characteristics over $f = f_{r-1} \circ f_{r-2} \circ \cdots \circ f_0$. A characteristic model is given by three bit-vector constraints: the validity constraint, the probability-one constraint and the weight constraint.

- The validity constraint with inputs $(\gamma_0, \gamma_1, \ldots, \gamma_r)$ is `True` if and only if the characteristic $\Gamma = (\gamma_0, \gamma_1 \ldots, \gamma_r)$ is valid.
- The probability-one constraint with inputs $(\gamma_0, \gamma_1, \ldots, \gamma_r)$ is `True` if and only if the propagation probability of the characteristic $\Gamma = (\gamma_0, \gamma_1 \ldots, \gamma_r)$ is 1.
- The weight constraint with inputs $(w, \gamma_0, \gamma_1, \ldots, \gamma_r)$ is `True` if and only if the bit-vector w is equal to the propagation weight of the valid characteristic $\Gamma = (\gamma_0, \gamma_1 \ldots, \gamma_r)$.

Given property models of the functions $f_0, f_1, \ldots$ and f_{r-1}, the constraints of the characteristic model are obtained as follows. Let $\mathrm{VC}_i, \mathrm{POC}_i$ and WC_i denote the validity, probability-one and weight constraint, respectively, of the property model of f_i. Then, the constraints $\mathrm{VC}, \mathrm{POC}$ and WC of the characteristic model are given by

$$\begin{aligned}
\mathrm{VC}(\gamma_0, \ldots, \gamma_r) &= \mathrm{VC}_0(\gamma_0, \gamma_1) \wedge \cdots \wedge \mathrm{VC}_{r-1}(\gamma_{r-1}, \gamma_r) \\
\mathrm{POC}(\gamma_0, \ldots, \gamma_r) &= \mathrm{POC}_0(\gamma_0, \gamma_1) \wedge \cdots \wedge \mathrm{POC}_{r-1}(\gamma_{r-1}, \gamma_r) \\
\mathrm{WC}(w, \gamma_0, \ldots, \gamma_r) &= \exists w_0, \ldots, w_{r-1} : (w = w_0 \boxplus \cdots \boxplus w_{r-1}) \wedge \\
&\qquad \mathrm{WC}_0(w_0, \gamma_0, \gamma_1) \wedge \cdots \wedge \mathrm{WC}_{r-1}(w_r, \gamma_{r-1}, \gamma_r)
\end{aligned}$$

Moreover, $\ell = \max(\ell_0, \ldots, \ell_{r-1})$ is the number of fractional bits and $\epsilon = \epsilon_0 + \cdots + \epsilon_{r-1}$ is the error bound of WC, where ℓ_i is the number of fractional bits and ϵ_i the error bound of WC_i. Note also that the bitsize of the weight variables in WC might need to be increased (by left concatenating with zeros) to avoid overflows in $w = w_0 \boxplus \cdots \boxplus w_{r-1}$.

4 Bit-vector Automated Methods

In this section we describe several systematic and automated methods to search for high-probability characteristics and zero-probability global properties by solving a sequence of bit-vector SMT problems. Our methods generalize the SMT-based search for differential characteristics of ARX ciphers by Mouha and Preneel [MP13], the automated search for impossible differentials of ciphers

with small S-boxes by Sasaki and Todo [ST17], and the SMT-based miss-in-the-middle search for related-key impossible differentials of ARX ciphers by Azimi et al. [Azi+22]. Moreover, we propose a new automated method to search for zero-probability global properties based of quantified bit-vector SMT problems.

These systematic methods can be applied for an arbitrary bit-vector property. In particular, for the properties previously defined (XOR difference, RX difference and linear mask properties), these systematic methods can be used to mount the following cipher attacks: (related-key) differential, RX, (related-key) impossible-differential, impossible-RX, linear and zero-correlation cryptanalysis.

While we focus here on block ciphers, it is worth to mention that these systematic methods can also be used to search for exploitable properties over other cryptographic primitives, as some of these distinguishing attacks have a counterpart for Message Authentication Code (MAC) algorithms or hash functions [BS93].

4.1 Search for Low-weight Characteristics

In this section we describe an automated method to search for low-weight characteristics for an arbitrary bit-vector property by solving a sequence of bit-vector SMT problems. This method generalizes the SMT-based method to search for differential characteristic of ARX ciphers by [MP13].

Let $(\mathrm{VC}, \mathrm{POC}, \mathrm{WC})$ be the constraints of a characteristic model of a function $f = f_{r-1} \circ f_{r-2} \circ \cdots \circ f_0$. Finding a characteristic with integer weight w can be done by solving with a bit-vector SMT solver the bit-vector SMT problem

$$\begin{aligned} &\exists \gamma_0, \gamma_1, \ldots, \gamma_r, w, w' : \\ &\quad \mathrm{VC}(\gamma_0, \gamma_1, \ldots, \gamma_r) \\ &\quad \mathrm{WC}(w', \gamma_0, \gamma_1, \ldots, \gamma_r) \\ &\quad w = \mathrm{Truncate}(w', \ell) \end{aligned} \tag{5}$$

where $\mathrm{Truncate}(w', \ell)$ ignores the ℓ least significant bits by extracting the $n'_w - \ell$ most significant bits.

To search for a characteristic with the lowest integer weight, the previous subroutine is simply repeated starting with integer weight $w = 0$ and incrementing the integer weight if the current SMT problem is unsatisfiable. If the error bound ϵ of the characteristic model is zero, the first satisfiable problem leads to an optimal characteristic, in the sense that there are no characteristics with integer weight strictly smaller, and the search finishes.

Otherwise, let $\hat{w}$ be the integer weight of the first characteristic obtained. The search finishes after all characteristics with integer weights in the interval $[\hat{w}, \hat{w}+\epsilon]$ are obtained, and the one with the lowest weight (an optimal characteristic) is returned. Note that given a characteristic $\Gamma' = (\gamma'_0, \gamma'_1, \ldots, \gamma'_r)$ with integer weight w, obtaining another characteristic with integer weight w can be done by solving the SMT problem given by Eqn. (5) with the additional constraint $(\gamma_0 \neq \gamma'_0) \vee (\gamma_1 \neq \gamma'_1) \vee \cdots \vee (\gamma_r \neq \gamma'_r)$, and this can be repeated to obtain all characteristics with integer weight w.

In practice, the search can be speeded up by first searching for an optimal characteristic Γ_0 over the simple function f_0 and then using the integer weight of Γ_0 as the starting weight of the search over $f_1 \circ f_0$; this process is iteratively repeated until $f = f_{r-1} \circ f_{r-2} \circ \cdots \circ f_0$. This iterative process exploits the fact that if all SMT problems for $f_i \circ f_{i-1} \circ \cdots \circ f_0$ and for integer weights $\{0, 1, \ldots, w\}$ were found unsatisfiable, then all SMT problems for $f_{i+1} \circ f_i \circ f_{i-1} \cdots \circ f_0$ and for integer weights $\{0, 1, \ldots, w\}$ are also unsatisfiable, as the characteristic weight is defined as the sum of the non-negative local propagation weights.

This automated method can be used to search for differential or linear characteristics of a block cipher simply by setting f as the encryption function E_k. Related-key differential characteristics can also be searched for simply by extending Eqn. (5) for a pair of characteristic models $(\Gamma_{\mathrm{KS}}, \Gamma_{E_k})$, and constraining the sum of the propagation weight of Γ_{KS} and the propagation weight of Γ_{E_k} to the target integer weight w. Moreover, additional constraints can be added to the SMT problems. For example, the constraint $\gamma_0 \neq 0 \neq \gamma_r$ can be added to exclude trivial characteristics, or the probability-one constraint of KS can be used (rather than the weight constraint) to search for related-key differential characteristics with key-schedule zero weight faster.

For some properties such as the difference properties, the propagation probability of a global property (α, β) can be estimated by summing the propagation probabilities of all characteristics with input property α and output property β. In this case, the probability of (α, β) can be estimated with this automated method by adding additional constraints fixing the input and output property of the characteristic and searching for all characteristics.

4.2 Search for Invalid Properties

In this section we explain how to search for zero-probability global properties by describing three bit-vector SMT-based methods: (1) the brute-force method generalizes the automated search for impossible differentials of ciphers with small S-boxes by [ST17], (2) the miss-in-the-middle method generalizes the search for related-key impossible differentials of ARX ciphers by [Azi+22], and (3) the quantified method is a new automated method based of quantified bit-vector SMT problems.

Brute-force method. Let VC be the validity constraint of a characteristic model of a function $f = f_{r-1} \circ f_{r-2} \circ \cdots \circ f_0$, and let (α, β) be a property of f for some bit-vector constants α and β. The brute-force and miss-in-the-middle methods are based on the fact that if the bit-vector SMT problem

$$\exists \gamma_0, \gamma_1, \ldots, \gamma_r : \mathrm{VC}(\gamma_0, \gamma_1, \ldots, \gamma_r) \wedge (\alpha = \gamma_0) \wedge (\beta = \gamma_r) \tag{6}$$

is unsatisfiable, then (α, β) has zero propagation probability. The difference between these two methods is the choice of the properties (α, β). The brute-force method simply chooses a subset of properties with many zero bits and checks whether the SMT problem given by Eqn. (6) is unsatisfiable for each property.

This choice is due to the fact that for some functions most of the impossible differentials found this far have many zero bits [ST17].

Miss-in-the-middle method. The idea of the miss-in-the-middle technique [BBS99a; BBS99b] is to find an impossible differential built from two probability-one characteristics Γ_0 and Γ_2, where the characteristic Γ_0 (resp. Γ_2) covers the first (resp. second) half of cipher, such that the output difference α of Γ_0 does not match the input difference β of Γ_2 in the middle of the cipher. For simplicity, our automated miss-in-the-middle method is explained for $f = f_2 \circ f_1 \circ f_0$; the generalization for $r \geq 3$ is straightforward.

Let $\mathrm{VC}_i(\gamma_i, \gamma_{i+1})$ and $\mathrm{POC}_i(\gamma_i, \gamma_{i+1})$ be the validity and probability-one constraints, respectively, of the characteristic model of f_i. First, a pair of probability-one characteristics (Γ_0, Γ_2) of f_0 and f_2 is found by solving the bit-vector SMT problem

$$\exists \gamma_0, \gamma_1, \gamma_2, \gamma_3 : \mathrm{POC}_0(\gamma_0, \gamma_1) \wedge \mathrm{POC}_2(\gamma_2, \gamma_3)\,. \tag{7}$$

Let (α', α) be the input and output properties of Γ_0 and (β, β') the input and output properties of Γ_2. Then, a similar problem than Eqn. (6) is built for the property (α, β) of f_1. If the problem is unsatisfiable, (α, β) is a zero-probability property of f_1, and by construction (α', β') is a zero-probability property of f. Otherwise, this process is repeated by finding another pair of probability-one characteristics of f_0 and f_2.

As the brute-force method, the miss-in-the-middle method is based on the unsatisfiability of SMT problems in the form of Eqn. (6), but where the properties (α, β) are chosen as the outputs and inputs of probability-one characteristics covering the initial and last part of the cipher respectively.

Quantified method. As opposed to the brute-force and the miss-in-the-middle methods, the quantified method is based on solving a satisfiable bit-vector SMT problem that combines existential and for-all quantifiers. Given the validity constraint VC of a characteristic model of $f = f_{r-1} \circ f_{r-2} \circ \cdots \circ f_0$, consider the quantified bit-vector SMT problem

$$\exists \gamma_0, \gamma_r, \forall \gamma_1, \gamma_2 \ldots, \gamma_{r-1} : \mathrm{VC}(\gamma_0, \gamma_1, \ldots, \gamma_r) = \texttt{False}\,. \tag{8}$$

If satisfiable, a solution of this problem is an assignment of the variables (γ_0, γ_r) such that the characteristic $\Gamma = (\gamma_0, \gamma_1, \ldots, \gamma_r)$ is invalid for all intermediate properties $(\gamma_1, \gamma_2, \ldots, \gamma_{r-1})$. In other words, a solution of this problem is a property (γ_0, γ_r) of f with zero propagation probability.

Thus, zero-probability properties of f can be obtained by solving the problem given by Eqn. (8) with an SMT solver supporting quantified bit-vector formulas such as Boolector or Z3 [MB08; WHM13].

Although quantifier-free problems, used by the brute-force and miss-in-the-middle methods, can be solved much faster than quantified problems, any zero-probability property found by the brute-force or the miss-in-the-middle method can be found by the quantified method, and the latter method can find zero-probability properties unreachable by the brute-force and the miss-in-the-middle methods.

These three automated methods can be used to search for impossible differentials or zero-correlation linear approximations of a block cipher simply by setting f as the encryption function E_k. Related-key impossible differentials can also be searched for with these three automated methods simply by extending the SMT problems given by Eqns. (6) to (8) to related-key differential characteristics.

Note that these three methods are sound but not complete methods; any property found by these methods has zero propagation probability, but some zero-probability properties might not be found by these methods. In other words, if the SMT problem given by Eqn. (6) is unsatisfiable, then (α, β) has zero propagation probability, but the other way around does not hold in general (and similarly for the SMT problem given by Eqn. (8)). In the single-key setting, these methods are complete if assuming that the round keys are independent, chosen uniformly at random, and XORed to the whole state before each non-linear operation [ST17].

5 The Tool `CASCADA`

In this section we describe the tool `CASCADA` [Ran22], an open-source Python library that implements the bit-vector property framework described in Sect. 3 and the bit-vector automated methods described in Sect. 4.

The tool `CASCADA` is based on `ArxPy` [Ran], a tool to search for differential characteristics and impossible differentials of ARX ciphers. However, while `ArxPy` restricts to (related-key) differential, RX, and (related-key) impossible-differential cryptanalysis, `CASCADA` implements the bit-vector property framework, new automated methods, and many new functionalities and improvements.

In particular, `CASCADA` implements the XOR difference, RX difference and linear mask, and new bit-vector properties can be easily added. Moreover, `CASCADA` implements the XOR differential, RX differential and linear models of many bit-vector operations, and it implements the weak and branch-based models and the property model based on weight tables[3]. As a result, `CASCADA` can search for (related-key) XOR differential, RX differential, and linear characteristics, and `CASCADA` can also search for (related-key) XOR impossible differentials, RX impossible differentials and zero-correlation linear approximations.

Compared to `ArxPy`, the cipher interface in `CASCADA` has been improved to support not only ARX ciphers but also other ciphers and primitives, and the documentation has been extended so that each Python function and class contains a detailed docstring with usage examples as doctests. On top of that, `CASCADA` includes a complete test suite for each functionality, and many unit tests follow the property-based testing technique [CH00] to test program properties on random inputs.

[3] For example, for ciphers with S-boxes, `CASCADA` can find the minimum number of active S-boxes by using the weak model for the S-boxes, but can also find full characteristics by using the model based on weight tables for the S-boxes.

The user workflow to run one of the automated search methods with CASCADA for a given primitive is the following. First, the user implements the primitive following the interface provided by CASCADA; the user can also choose one of the many primitives already implemented. Then, if the property model of an operation of the primitive is not provided by CASCADA, the user can either implement the property model or simply use a weak, branch-based or table-based model. Finally, the user chooses the search method and its parameters (e.g., the bit-vector property, the SMT solver, additional constraints, etc.), and starts the search.

In the search, CASCADA generates the characteristic model from the Python implementation of the primitive, encodes the SMT problems, and solves the SMT problems by querying an external SMT solver. These steps, depicted in Figure 1, are performed by CASCADA internally. Thus, using CASCADA does not require any knowledge about SMT problems or SMT solvers as this is automatically handled by CASCADA. Note that the running time of the search is dominated by the time the SMT solver takes to solve the SMT problems, and the steps performed by CASCADA introduce negligible overhead.

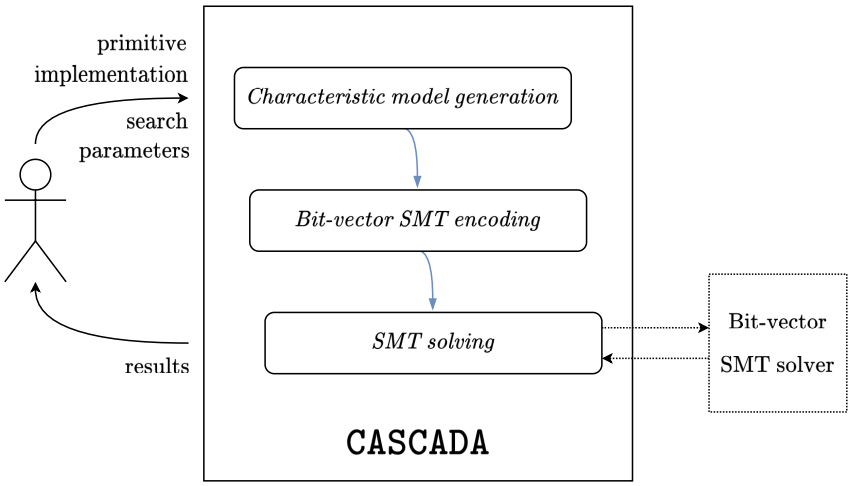

Figure 1. Main steps performed by CASCADA in an automated search method.

The library CASCADA has a modular and loose-coupling design split in several modules, namely the bit-vector module, the primitive module, the property modules and the SMT module, so that each module can be used and extended independently. The rest of this section explains a high-level overview of the implementation and functionality of each module, and a full description of each module can be found in the documentation of CASCADA.

5.1 Bit-Vector Module

The bit-vector module handles the creation, evaluation, symbolic manipulation and representation of bit-vector expressions and functions. To this end, it provides data types to create bit-vector constants, variables, operations, expressions and functions, it relies on `SymPy` [Meu+17] (an open-source Python library for symbolic computation) for the bit-vector symbolic manipulation, and it provides several representations of the bit-vector data types including an executable string representation, a C code representation or a DOT (a graph description language) representation.

To create bit-vector expressions, the bit-vector operations described in Sect. 2.1, which are also the bit-vector operations supported by the SMT-LIB Standard 2.0 [BST10], are implemented in the bit-vector module and called the primary operations. This module also implements other bit-vector operations such as the bit-wise majority, the bit-wise conditional, the bit-reversal or the hamming weight, and it supports bit-vector operations given by look-up tables or binary matrices. All non-primary operations are implemented as bit-vector expressions of primary operations so that they can be easily represented in bit-vector SMT problems; the hamming weight and bit-reversal are efficiently implemented as bit-vector expressions by using a divide-and-conquer approach from [HSW03].

The bit-vector module also provides several context managers to modify the creation, evaluation and manipulation of bit-vector expressions. For example, the simplification context controls whether to simplify expressions by applying Boolean algebra rules, and the memoization context is a space-time trade-off also known as tabling where intermediate results are stored in a table so that they can be retrieved when the same inputs occur again.

Inspired by the representation of mathematical functions in `SymPy`, the bit-vector module provides a similar interface to represent bit-vector functions with or without external values; round-based functions are also supported and even non-bit-vector functions by using undefined bit-vector operations. A bit-vector function can be converted into a (bit-vector) Static Single Assignment (SSA) [RWZ88] object, that is, a list of assignments where each instruction is a bit-vector operation and each variable is assigned exactly once and defined before used. The bit-vector module also implements decomposing an SSA object of a round-based function into the SSA objects of its rounds, representing the graph of an SSA object in the DOT language, and translating, compiling and evaluating an SSA object into a C executable.

It is worth to mention that the bit-vector module does not depend on other modules of `CASCADA`, and thus it can be used independently in applications requiring the symbolic manipulation of bit-vector expressions or functions.

5.2 Primitive Module

The primitive module provides data types to represent encryption functions and block ciphers. While key-schedule functions can be implemented directly as bit-vector functions, the encryption function type specifies a bit-vector function and

a list of round keys, and the block cipher type specifies a key-schedule function and an encryption function.

The primitive module implements many cryptographic primitives, namely AES [Dwo+01], a masked AES [Bey+21], CHAM [Koo+17], Chaskey [Mou+14], FEAL [Miy90], HIGHT [Hon+06], LEA [Hon+13], MULTI2 [ISO94], π-cipher [Gli+14], SHACAL-1 [HKR01], SHACAL-2 [HN02], Simeck [Yan+15], Simon [Bea+15], Speck [Bea+15], SKINNY [Bei+16], TEA [WN94], and XTEA [NW97].

5.3 Property Modules

The property modules of `CASCADA` consist of the abstract property module, providing the interface to implement bit-vector properties, and the differential and linear modules, which instantiate the abstract property module for the difference and linear mask properties respectively.[4] Most of the logic and functionality is implemented in the abstract property module so that new bit-vector properties can be easily added.

The abstract property module provides the data types to represent bit-vector properties, property models, characteristics and characteristic models. In particular, it implements the weak model, the branch-based model, and the property model based on weight tables, and it provides three characteristic data types to represent (1) characteristics over bit-vector functions, (2) characteristics over encryption functions, and (3) pairs of characteristics over key-schedule and encryption functions.

The abstract property module also implements the generation of characteristic models, the decomposition of characteristics and characteristic models of round-based functions, the DOT representation of characteristics and characteristic models and the computation of empirical weights.

Given a characteristic $\Gamma = (\gamma_0, \gamma_1, \ldots, \gamma_{r-1})$ of a bit-vector function f, the empirical weight is defined in `CASCADA` as an estimation of the propagation weight of (γ_0, γ_{r-1}) computed by evaluating f for many inputs satisfying γ_0. Since for some properties the number of inputs required to obtain a meaningful estimation is exponential in the propagation weight of the characteristic, the computation of the empirical weight automatically splits a high-weight characteristic into low-weight characteristics and evaluates f by translating and compiling the SSA of f to a C executable.

The differential module instantiates the abstract property module for the XOR difference and RX difference properties. It implements the XOR and RX trivial models of many $\oplus$-linear and propagation-deterministic operations, that is, operations that propagate an input property to a unique output property with probability one.

For the XOR difference property, the differential module implements the non-trivial models of the following operations: the unary operators $\boxplus_c(x) = x \boxplus c$

[4] An additional property module, the algebraic module, is also implemented in `CASCADA` instantiating the abstract property module for another bit-vector property, the value property.

and $\boxminus_c(x) = x \boxminus c$, the binary operators $\wedge, \vee, \boxplus, \boxminus$, and the ternary operators bit-wise majority, bit-wise conditional and the Simon round function $f_{a,b,c}$. We implemented the models of $\boxplus_c, \boxplus$ and $f_{a,b,c}$ from previous work [Azi+20; LM01; KLT15], we derived the models for $\boxminus_c$ and $\boxminus$ from the models of the modular addition by using the identity $\neg(x \boxminus y) = \neg x \boxplus y$ [HSW03], and we derived the models of the bit-wise operations ($\wedge, \vee$, bit-wise majority and bit-wise conditional) by extrapolating the constraints for 1-bit inputs.

For the RX difference property, the differential module implements the non-trivial models of the unary operators $\ll_c (x) = x \ll c$ and $\gg_c (x) = x \gg c$, the binary operators $\wedge, \vee, \boxplus, \boxminus$, and the ternary operators bit-wise majority, bit-wise conditional and the Simon round function $f_{a,b,c}$. The models of $\ll_c$ and $\gg_c$ were based on the rotational analysis from [Sok16], and the model of $\boxminus$ was derived from the model of $\boxplus$ [AL16] and the identity $\neg(x \boxminus y) = \neg x \boxplus y$. The rest of the models are derived from their XOR models since an XOR model of a function f commuting with $\lll_1$ is also an RX model of f.

The linear module instantiates the abstract property module for the linear mask property. Apart from the trivial models of $\oplus$-linear and propagation-deterministic operations, it implements the non-trivial models of $\boxplus$ [Sch13], $\boxminus$ (from the $\boxplus$ model and the identity $\neg(x \boxminus y) = \neg x \boxplus y$), and of $\wedge$ and $\vee$ by extrapolating the constraints for 1-bit inputs.

The generation of linear characteristic models automatically handles the branches in SSA objects. An input linear mask α can propagate through a branch $x \mapsto (x, x)$ to multiple output masks (i.e., to any output mask (β_0, β_1) such that $\alpha \oplus \beta_0 \oplus \beta_1 = 0$), and branches are automatically detected and handled whenever a variable x is used multiple times in an SSA object.

For any property, the generation of a characteristic model of a bit-vector function f in `CASCADA` is performed as described in Sect. 3.3, where the property models of the bit-vector operations in the SSA of f are used to build the constraints of the characteristic model. Thus, characteristic models can be directly generated for any function f composed of bit-vector operations with property models implemented in `CASCADA`. For functions including operations without models in `CASCADA`, one can fully implement their models (if their property models are known) or use weak, branch-based or table-based models which can be easily obtained for any function in `CASCADA`.

5.4 SMT Module

The SMT module implements the automated methods described in Sect. 4. To solve the underlying SMT problems, the SMT module relies on `PySMT` [GM15], an open-source Python API for SMT solvers. As a result, the SMT module supports any of the bit-vector SMT solvers natively supported by `PySMT` (i.e, Boolector, CVC4 [Bar+11], MathSAT [Cim+13], Z3, and Yices [Dut14]), and it can also use other solvers through the interface of `PySMT`.

Apart from the choice of the bit-vector SMT solver, many options in the automated methods implemented in the SMT module can be configured, including

the type of constraints (e.g., validity and weight constraints or only probability-one constraints), additional constraints, the verbose level, or whether to filter characteristics using the empirical weight. If this last option is enabled, after a characteristic is obtained as a solution of an SMT problem, the empirical weight of the characteristic is computed, and characteristics with large approximation errors between their propagation weights and their empirical weights are discarded.

Most of the automated methods of Sect. 4 require solving a sequence of SMT problems built incrementally from a base SMT problem. For example, this occurs when multiple solutions are required from an SMT problem, or in the search for characteristics of round-based functions. This type of incremental queries are common to SMT solvers, and many of them support an incremental mode [BT18], where computations from previous problems are reused to solve the next query. The SMT module implements this type of sequences of SMT problems as incremental queries, leveraging the incrementality feature of SMT solvers.

As in the other modules, we implemented an extensive suite of tests in the SMT module. In particular, we tested the search for low-weight characteristics in the primitives implemented in `CASCADA` by revisiting the following previous work listing the weights of optimal characteristics covering small number of rounds. This includes XOR differential and linear characteristics of AES [Mou+11, Table 4], XOR differential and linear characteristics of CHAM, [Roh+19, Table 10, Table 11], linear characteristics of Chaskey [LWR16, Table 4], XOR differential and linear characteristics of HIGHT [Yin+17, Table 4, Table 7], XOR [KLT15, Table 1], related-key XOR [Wan+18, Table 5] and RX [Lu+20, Table 5] differential characteristics of Simeck, XOR [LLW17, Table 1] related-key XOR [Wan+18, Table 5] and RX [Lu+20, Table 5] differential characteristics of Simon, XOR differential and linear characteristics of SKINNY [Bei+16, Table 7] and XOR differential [BVC16, Table 4] and linear [LWR16, Table 2] characteristics of Speck.

6 Conclusion and Future Work

In this work we presented the tool `CASCADA`, and we described the bit-vector property framework and the automated methods implemented in `CASCADA`, including the new automated method based on quantified problems. Moreover, we provided a high-level overview of the functionality and implementation of the modules in `CASCADA`. This overview is not exhaustive and a complete description of the functionality and features of `CASCADA` can be found in its documentation.

The tool `CASCADA` not only aims to facilitate designers and cryptanalysts the security evaluation of cryptographic primitives but also to assist further research in automated methods. For example, no property models have been researched for many operations, such as a linear model of the modular addition by a constant $x \mapsto x \boxplus c$ or a differential model of the rotation by a variable $(x, y) \mapsto x \lll y$, and if new models are researched, they can be easily implemented and tested in `CASCADA`. Similarly, no bit-vector automated method has been proposed for several distinguishing attacks, such as truncated differential [Knu94] or linear

cryptanalysis in the related-key setting [Bog+13], and new bit-vector properties can also be easily implemented and tested in `CASCADA`.

Acknowledgements. Adrián Ranea is supported by a PhD Fellowship from the Research Foundation – Flanders (FWO) with grant number 11E1921N. The authors would like to thank the anonymous reviewers for their comments and suggestions.

References

[Abd+17] Ahmed Abdelkhalek et al. 'MILP Modeling for (Large) S-boxes to Optimize Probability of Differential Characteristics'. In: *IACR Trans. Symmetric Cryptol.* 2017.4 (2017), pp. 99–129.

[ABR20] Tomer Ashur, Tim Beyne and Vincent Rijmen. 'Revisiting the Wrong-Key-Randomization Hypothesis'. In: *J. Cryptol.* 33.2 (2020), pp. 567–594.

[ÁK16] Erika Ábrahám and Gereon Kremer. 'Satisfiability Checking: Theory and Applications'. In: *SEFM*. Vol. 9763. Lecture Notes in Computer Science. Springer, 2016, pp. 9–23.

[AK18] Ralph Ankele and Stefan Kölbl. 'Mind the Gap - A Closer Look at the Security of Block Ciphers against Differential Cryptanalysis'. In: *SAC*. Vol. 11349. Lecture Notes in Computer Science. Springer, 2018, pp. 163–190.

[AL16] Tomer Ashur and Yunwen Liu. 'Rotational Cryptanalysis in the Presence of Constants'. In: *IACR Trans. Symmetric Cryptol.* 2016.1 (2016), pp. 57–70.

[AL18] Ralph Ankele and Eik List. 'Differential Cryptanalysis of Round-Reduced Sparx-64/128'. In: *ACNS*. Vol. 10892. Lecture Notes in Computer Science. Springer, 2018, pp. 459–475.

[Azi+20] Seyyed Arash Azimi et al. 'A Bit-Vector Differential Model for the Modular Addition by a Constant'. In: *ASIACRYPT (1)*. Vol. 12491. Lecture Notes in Computer Science. Springer, 2020, pp. 385–414.

[Azi+22] Seyyed Arash Azimi et al. 'A Bit-Vector Differential Model for the Modular Addition by a Constant and its Applications to Differential and Impossible-Differential Cryptanalysis'. In: *IACR Cryptol. ePrint Arch.* (2022).

[Bar+11] Clark W. Barrett et al. 'CVC4'. In: *CAV*. Vol. 6806. Lecture Notes in Computer Science. Springer, 2011, pp. 171–177.

[BBS99a] Eli Biham, Alex Biryukov and Adi Shamir. 'Cryptanalysis of Skipjack Reduced to 31 Rounds Using Impossible Differentials'. In: *EUROCRYPT*. Vol. 1592. Lecture Notes in Computer Science. Springer, 1999, pp. 12–23.

[BBS99b] Eli Biham, Alex Biryukov and Adi Shamir. 'Miss in the Middle Attacks on IDEA and Khufu'. In: *FSE*. Vol. 1636. Lecture Notes in Computer Science. Springer, 1999, pp. 124–138.

[Bea+15] Ray Beaulieu et al. 'The SIMON and SPECK lightweight block ciphers'. In: *DAC*. ACM, 2015, 175:1–175:6.

[Bei+16] Christof Beierle et al. 'The SKINNY Family of Block Ciphers and Its Low-Latency Variant MANTIS'. In: *CRYPTO (2)*. Vol. 9815. Lecture Notes in Computer Science. Springer, 2016, pp. 123–153.

[Bey+21] Tim Beyne et al. 'A Low-Randomness Second-Order Masked AES'. In: *SAC*. Vol. 13203. Lecture Notes in Computer Science. Springer, 2021, pp. 87–110.

[Bih94] Eli Biham. 'New Types of Cryptanalytic Attacks Using Related Keys'. In: *J. Cryptol.* 7.4 (1994), pp. 229–246.

[Bog+13] Andrey Bogdanov et al. 'Key Difference Invariant Bias in Block Ciphers'. In: *ASIACRYPT (1)*. Vol. 8269. Lecture Notes in Computer Science. Springer, 2013, pp. 357–376.

[Bou+18] Christina Boura et al. 'Making the Impossible Possible'. In: *J. Cryptol.* 31.1 (2018), pp. 101–133.

[BR14] Andrey Bogdanov and Vincent Rijmen. 'Linear hulls with correlation zero and linear cryptanalysis of block ciphers'. In: *Des. Codes Cryptogr.* 70.3 (2014), pp. 369–383.

[BS93] Eli Biham and Adi Shamir. *Differential Cryptanalysis of the Data Encryption Standard*. Springer, 1993.

[BST10] Clark Barrett, Aaron Stump and Cesare Tinelli. 'The SMT-LIB Standard: Version 2.0'. In: *Proceedings of the 8th International Workshop on Satisfiability Modulo Theories (Edinburgh, UK)*. Ed. by A. Gupta and D. Kroening. 2010.

[BT13] Andrey Bogdanov and Elmar Tischhauser. 'On the Wrong Key Randomisation and Key Equivalence Hypotheses in Matsui's Algorithm 2'. In: *FSE*. Vol. 8424. Lecture Notes in Computer Science. Springer, 2013, pp. 19–38.

[BT18] Clark Barrett and Cesare Tinelli. 'Satisfiability Modulo Theories'. In: *Handbook of Model Checking*. Springer, 2018, pp. 305–343.

[BVC16] Alex Biryukov, Vesselin Velichkov and Yann Le Corre. 'Automatic Search for the Best Trails in ARX: Application to Block Cipher Speck'. In: *FSE*. Vol. 9783. Lecture Notes in Computer Science. Springer, 2016, pp. 289–310.

[BW12] Andrey Bogdanov and Meiqin Wang. 'Zero Correlation Linear Cryptanalysis with Reduced Data Complexity'. In: *FSE*. Vol. 7549. Lecture Notes in Computer Science. Springer, 2012, pp. 29–48.

[CH00] Koen Claessen and John Hughes. 'QuickCheck: a lightweight tool for random testing of Haskell programs'. In: *ICFP*. ACM, 2000, pp. 268–279.

[Cim+13] Alessandro Cimatti et al. 'The MathSAT5 SMT Solver'. In: *TACAS*. Vol. 7795. Lecture Notes in Computer Science. Springer, 2013, pp. 93–107.

[DR07] Joan Daemen and Vincent Rijmen. 'Probability distributions of correlation and differentials in block ciphers'. In: *J. Math. Cryptol.* 1.3 (2007), pp. 221–242.

[Dut14] Bruno Dutertre. 'Yices 2.2'. In: *CAV*. Vol. 8559. Lecture Notes in Computer Science. Springer, 2014, pp. 737–744.

[Dwo+01] Morris Dworkin et al. *Advanced Encryption Standard (AES)*. en. 2001.

[GD07] Vijay Ganesh and David L. Dill. 'A Decision Procedure for Bit-Vectors and Arrays'. In: *CAV*. Vol. 4590. Lecture Notes in Computer Science. Springer, 2007, pp. 519–531.

[Gli+14] Danilo Gligoroski et al. 'π-Cipher: Authenticated Encryption for Big Data'. In: *NordSec*. Vol. 8788. Lecture Notes in Computer Science. Springer, 2014, pp. 110–128.

[GM15] Marco Gario and Andrea Micheli. 'PySMT: a solver-agnostic library for fast prototyping of SMT-based algorithms'. In: *SMT Workshop 2015*. 2015.

[Had+19] Hosein Hadipour et al. 'Comprehensive security analysis of CRAFT'. In: *IACR Trans. Symmetric Cryptol.* 2019.4 (2019), pp. 290–317.

[HKR01] Helena Handschuh, Lars R. Knudsen and Matthew J. B. Robshaw. 'Analysis of SHA-1 in Encryption Mode'. In: *CT-RSA*. Vol. 2020. Lecture Notes in Computer Science. Springer, 2001, pp. 70–83.

[HN02] Helena Handschuh and David Naccache. 'SHACAL: a family of block ciphers'. In: *Submission to the NESSIE project* (2002).

[Hon+06] Deukjo Hong et al. 'HIGHT: A New Block Cipher Suitable for Low-Resource Device'. In: *CHES*. Vol. 4249. Lecture Notes in Computer Science. Springer, 2006, pp. 46–59.

[Hon+13] Deukjo Hong et al. 'LEA: A 128-Bit Block Cipher for Fast Encryption on Common Processors'. In: *WISA*. Vol. 8267. Lecture Notes in Computer Science. Springer, 2013, pp. 3–27.

[HSW03] Jr. Henry S. Warren. *Hacker's delight*. Addison-Wesley, 2003.

[ISO94] ISO. *Algorithm Registry Entry 9979/0009*. 1994.

[KLT15] Stefan Kölbl, Gregor Leander and Tyge Tiessen. 'Observations on the SIMON Block Cipher Family'. In: *CRYPTO (1)*. Vol. 9215. Lecture Notes in Computer Science. Springer, 2015, pp. 161–185.

[Knu94] Lars R. Knudsen. 'Truncated and Higher Order Differentials'. In: *FSE*. Vol. 1008. Lecture Notes in Computer Science. Springer, 1994, pp. 196–211.

[Koo+17] Bonwook Koo et al. 'CHAM: A Family of Lightweight Block Ciphers for Resource-Constrained Devices'. In: *ICISC*. Vol. 10779. Lecture Notes in Computer Science. Springer, 2017, pp. 3–25.

[KPR20] Liliya Kraleva, Raluca Posteuca and Vincent Rijmen. 'Cryptanalysis of the Permutation Based Algorithm SpoC'. In: *INDOCRYPT*. Vol. 12578. Lecture Notes in Computer Science. Springer, 2020, pp. 273–293.

[LLW17] Zhengbin Liu, Yongqiang Li and Mingsheng Wang. 'Optimal Differential Trails in SIMON-like Ciphers'. In: *IACR Trans. Symmetric Cryptol.* 2017.1 (2017), pp. 358–379.

[LM01] Helger Lipmaa and Shiho Moriai. 'Efficient Algorithms for Computing Differential Properties of Addition'. In: *FSE*. Vol. 2355. Lecture Notes in Computer Science. Springer, 2001, pp. 336–350.

[Lod10] Andrea Lodi. 'Mixed Integer Programming Computation'. In: *50 Years of Integer Programming*. Springer, 2010, pp. 619–645.

[Lu+20] Jinyu Lu et al. 'Rotational-XOR Cryptanalysis of Simon-Like Block Ciphers'. In: *ACISP*. Vol. 12248. Lecture Notes in Computer Science. Springer, 2020, pp. 105–124.

[LWR16] Yunwen Liu, Qingju Wang and Vincent Rijmen. 'Automatic Search of Linear Trails in ARX with Applications to SPECK and Chaskey'. In: *ACNS*. Vol. 9696. Lecture Notes in Computer Science. Springer, 2016, pp. 485–499.

[Mat93] Mitsuru Matsui. 'Linear Cryptanalysis Method for DES Cipher'. In: *EUROCRYPT*. Vol. 765. Lecture Notes in Computer Science. Springer, 1993, pp. 386–397.

[MB08] Leonardo Mendonça de Moura and Nikolaj Bjørner. 'Z3: An Efficient SMT Solver'. In: *TACAS*. Vol. 4963. Lecture Notes in Computer Science. Springer, 2008, pp. 337–340.

[MB11] Leonardo Mendonça de Moura and Nikolaj Bjørner. 'Satisfiability modulo theories: introduction and applications'. In: *Commun. ACM* 54.9 (2011), pp. 69–77.

[Meu+17] Aaron Meurer et al. 'SymPy: symbolic computing in Python'. In: *PeerJ Comput. Sci.* 3 (2017), e103.

[Miy90] Shoji Miyaguchi. 'The FEAL Cipher Family'. In: *CRYPTO*. Vol. 537. Lecture Notes in Computer Science. Springer, 1990, pp. 627–638.

[Mou+11] Nicky Mouha et al. 'Differential and Linear Cryptanalysis Using Mixed-Integer Linear Programming'. In: *Inscrypt*. Vol. 7537. Lecture Notes in Computer Science. Springer, 2011, pp. 57–76.

[Mou+14] Nicky Mouha et al. 'Chaskey: An Efficient MAC Algorithm for 32-bit Microcontrollers'. In: *Selected Areas in Cryptography*. Vol. 8781. Lecture Notes in Computer Science. Springer, 2014, pp. 306–323.

[MP13] Nicky Mouha and Bart Preneel. 'A Proof that the ARX Cipher Salsa20 is Secure against Differential Cryptanalysis'. In: *IACR Cryptol. ePrint Arch.* (2013), p. 328.

[NPB14] Aina Niemetz, Mathias Preiner and Armin Biere. 'Boolector 2.0'. In: *J. Satisf. Boolean Model. Comput.* 9.1 (2014), pp. 53–58.

[NW97] Roger Needham and David Wheeler. *Tea extensions*. Tech. rep. Computer Laboratory, University of Cambridge, 1997.

[PS06] Raphael C.-W. Phan and Mohammad Umar Siddiqi. 'A Framework for Describing Block Cipher Cryptanalysis'. In: *IEEE Trans. Computers* 55.11 (2006), pp. 1402–1409.

[Ran] Adrián Ranea. *ArxPy: Tool to find XOR differential and rotational-XOR characteristics of ARX primitives.* https://github.com/ranea/ArxPy.

[Ran22] Adrián Ranea. *CASCADA*. Version v1.0.0. Apr. 2022. DOI: 10.5281/zenodo.6504337. URL: https://github.com/ranea/CASCADA.

[Rey16] Andrew Reynolds. 'Conflicts, Models and Heuristics for Quantifier Instantiation in SMT'. In: *Vampire@IJCAR*. Vol. 44. EPiC Series in Computing. EasyChair, 2016, pp. 1–15.

[RLA17] Adrián Ranea, Yunwen Liu and Tomer Ashur. 'An Easy-to-Use Tool for Rotational-XOR Cryptanalysis of ARX Block Ciphers'. In: *Proceedings of the Romanian Academy, Series A* 18.3 (2017).

[Roh+19] Dongyoung Roh et al. 'Revised Version of Block Cipher CHAM'. In: *ICISC*. Vol. 11975. Lecture Notes in Computer Science. Springer, 2019, pp. 1–19.

[RWZ88] Barry K. Rosen, Mark N. Wegman and F. Kenneth Zadeck. 'Global Value Numbers and Redundant Computations'. In: *POPL*. ACM Press, 1988, pp. 12–27.

[Sch13] Ernst Schulte-Geers. 'On CCZ-equivalence of addition mod 2 n'. In: *Des. Codes Cryptogr.* 66.1-3 (2013), pp. 111–127.

[SHY16] Ling Song, Zhangjie Huang and Qianqian Yang. 'Automatic Differential Analysis of ARX Block Ciphers with Application to SPECK and LEA'. In: *ACISP (2)*. Vol. 9723. Lecture Notes in Computer Science. Springer, 2016, pp. 379–394.

[Sok16] Przemysław Sokołowski. 'Design and Analysis of Cryptographic Hash Functions'. PhD thesis. Adam Mickiewicz University, Poland, 2016.

[ST17] Yu Sasaki and Yosuke Todo. 'New Impossible Differential Search Tool from Design and Cryptanalysis Aspects - Revealing Structural Properties of Several Ciphers'. In: *EUROCRYPT (3)*. Vol. 10212. Lecture Notes in Computer Science. 2017, pp. 185–215.

[Ste] Stefan Kölbl. *CryptoSMT: An easy to use tool for cryptanalysis of symmetric primitives.* https://github.com/kste/cryptosmt.

[Sun+14] Siwei Sun et al. 'Automatic Security Evaluation and (Related-key) Differential Characteristic Search: Application to SIMON, PRESENT, LBlock, DES(L) and Other Bit-Oriented Block Ciphers'. In: *ASIACRYPT (1)*. Vol. 8873. Lecture Notes in Computer Science. Springer, 2014, pp. 158–178.

[SWW18] Ling Sun, Wei Wang and Meiqin Wang. 'More Accurate Differential Properties of LED64 and Midori64'. In: *IACR Trans. Symmetric Cryptol.* 2018.3 (2018), pp. 93–123.

[Wag04] David A. Wagner. 'Towards a Unifying View of Block Cipher Cryptanalysis'. In: *FSE*. Vol. 3017. Lecture Notes in Computer Science. Springer, 2004, pp. 16–33.

[Wan+18] Xuzi Wang et al. 'Automatic Search for Related-Key Differential Trails in SIMON-like Block Ciphers Based on MILP'. In: *ISC*.

Vol. 11060. Lecture Notes in Computer Science. Springer, 2018, pp. 116–131.

[WHM13] Christoph M. Wintersteiger, Youssef Hamadi and Leonardo Mendonça de Moura. 'Efficiently solving quantified bit-vector formulas'. In: *Formal Methods Syst. Des.* 42.1 (2013), pp. 3–23.

[WN94] David J. Wheeler and Roger M. Needham. 'TEA, a Tiny Encryption Algorithm'. In: *FSE*. Vol. 1008. Lecture Notes in Computer Science. Springer, 1994, pp. 363–366.

[Yan+15] Gangqiang Yang et al. 'The Simeck Family of Lightweight Block Ciphers'. In: *CHES*. Vol. 9293. Lecture Notes in Computer Science. Springer, 2015, pp. 307–329.

[Yin+17] Jun Yin et al. 'Improved Cryptanalysis of an ISO Standard Lightweight Block Cipher with Refined MILP Modelling'. In: *Inscrypt*. Vol. 10726. Lecture Notes in Computer Science. Springer, 2017, pp. 404–426.