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

Adrián Ranea and Vincent Rijmen
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, especially 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 [AK16; 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 $b_0 + 2 b_1 + \cdots + 2^{n-1} b_{n-1}$. 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 $\land$, disjunction $\lor$, 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 $\bowtie$, modular subtraction $\boxminus$, modular multiplication, $\boxtimes$ unsigned truncated division operation $\boxdot$ and unsigned remainder (modulus) operation $\mod$.
- The relational operations: $=, <, >, \leq$ and $\geq$.
- The if-then-else operator 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
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 \( f_k(x, y) = (((x \gg 7) \oplus y) \oplus k, (y \ll 2) \oplus ((((x \gg 7) \oplus y) \oplus k)) \) 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

\[
\exists x, y, k, x', y' \in \{0, 1\}^{16}:
\begin{align*}
x' &= ((x \gg 7) \oplus y) \oplus k \\
y' &= (y \ll 2) \oplus x' \\
0 &= x \lor y \lor x' \lor y'.
\end{align*}
\]

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 \( k \in K \), 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 $\text{KS}(k) = (k_0, k_1, \ldots, k_{r-1})$ 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 $(\alpha, \beta)$ over the encryption function $E_k$ with high expected differential probability. Given a differential $(\alpha, \beta)$ over $f$, its differential probability is given by

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

where usually $\triangle = \oplus = \triangledown$. The operator $\triangledown$ computes the difference of a pair of values $(x, x')$, and the operator $\triangle$ takes as inputs a value $x$ and a difference $\alpha$ and outputs the value $x'$ such that the pair $(x, x')$ has difference $\alpha$.

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

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

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

$$\#\{x : E_k \triangle \kappa (x \triangle \alpha) \triangledown E_k(x) = \beta\} / 2^n.$$

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 $((\alpha, \gamma), (\beta, \gamma'))$ with high expected RX probability, where the RX probability over an $n$-bit function $f$ is defined as

$$\#\{x : f((x \ll \gamma) \oplus \alpha) \oplus (f(x) \ll \gamma') = \beta\} / 2^n.$$
The rotational offset $\gamma$ is usually fixed to 1, and in this case the RX difference pair $((\alpha, 1), (\beta, 1))$ is equivalent to the differential $(\alpha, \beta)$ with $\{\nabla, \triangle\}$ defined as

$$x' \nabla x = x' \oplus (x \ll 1), \quad x \triangle \alpha = (x \ll 1) \oplus \alpha .$$  \hspace{1cm} (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 $(\alpha, \beta)$ over the encryption function $E_k$ with high expected linear probability or potential. Let $C_f(\alpha, \beta)$ be the correlation of $(\alpha, \beta)$ over an $n$-bit function $f$ defined as

$$C_f(\alpha, \beta) = 2 \times \left( \# \{ x : \langle \alpha, x \rangle = \langle \beta, f(x) \rangle \} / 2^n \right) - 1,$$ \hspace{1cm} (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 $K$,

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

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 $(\alpha, \beta)$ with an associated propagation probability $PP_f(\alpha, \beta) \in [0, 1] \subseteq \mathbb{R}$. In this case, we

\(^1\) Follow-up studies of [Mat93] (e.g., [BT13; ABR20]) provide a more accurate estimation of the complexity of linear cryptanalysis.
also say that the input property $\alpha$ propagates to the output property $\beta$ with probability $\text{PP}_f(\alpha, \beta)$.

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

We consider here three types of properties$^2$: the XOR difference property, the RX difference property, and the linear mask property. A difference property $(\alpha, \beta)$ over a function $f$ is defined as the bit-vector property $(\alpha, \beta)$ 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 $\Delta = \oplus$ and RX difference properties consider $\Delta$ and $\nabla$ given by Eqn. (3).

A linear mask property $(\alpha, \beta)$ over $f$ is a bit-vector property $(\alpha, \beta)$ 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 $(\alpha, \beta)$ 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 $\Gamma$ over $f = f_{r-1} \circ f_{r-2} \circ \cdots \circ f_0$ is a trail of properties $(\gamma_0, \gamma_1, \ldots, \gamma_r)$ where $(\gamma_i, \gamma_{i+1})$ is a bit-vector property over $f_i$. The bit-vectors $(\gamma_0, \gamma_r)$ are also called the input and output property respectively of $\Gamma$. The propagation probability of $\Gamma$ is defined as

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

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 $(\Gamma_{KS}, \Gamma_{E_k})$ with $\Gamma_{KS}$ defined over the key-scheduling function $KS$ and $\Gamma_{E_k}$ over the encryption function $E_k$ such that the (external) round key properties of $\Gamma_{E_k}$ are set to the properties of $\Gamma_{KS}$. Depending on the function and the property, the propagation probability of a characteristic $\Gamma = (\gamma_0, \gamma_1, \ldots, \gamma_r)$ might not accurately approximate the propagation probability of the global property $(\gamma_0, \gamma_r)$, 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.

---

$^2$ The tool CASCADA also implements a fourth property type, the value property, where the propagation probability of the value property $(\alpha, \beta)$ 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.
In practice block ciphers are claimed secure against distinguishing attacks by showing that no high-probability characteristics and zero-probability global 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 \((\alpha, \beta)\) over a function \(f\) is valid if its propagation probability is non-zero. In this case, we define the propagation weight of \((\alpha, \beta)\) as the negative binary logarithm of its propagation probability, that is,

\[
PW_f(\alpha, \beta) = -\log_2(PP_f(\alpha, \beta)).
\]

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 \((\alpha, \beta)\) is \textbf{True} if and only if the property \((\alpha, \beta)\) is valid.
- The probability-one constraint with inputs \((\alpha, \beta)\) is \textbf{True} if and only if the propagation probability of the property \((\alpha, \beta)\) is 1.
- The weight constraint with inputs \((w, \alpha, \beta)\) is \textbf{True} if and only if the bit-vector \(w\) is equal to the propagation weight of the property \((\alpha, \beta)\).

The weight constraint is only defined for inputs \((\alpha, \beta)\) with non-zero propagation probability; the truth value of the weight constraint for invalid \((\alpha, \beta)\) 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 \((p, \alpha, \beta)\) being \textbf{True} if \(p\) is equal to the propagation probability of \((\alpha, \beta))\). 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 $2^{-\ell}(w[0] + 2w[1] + \cdots + 2^{n-1}w[n-1])$ for a given fixed number $\ell$ of fractional bits. Moreover, we also consider weight constraints that are true if and only if $|w - PW_f(\alpha, \beta)| < \epsilon$ for a fixed error bound $\epsilon$.

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 $f(x, x') = x \oplus x'$ were implicitly obtained in [LM01; Sch13], and an XOR differential model of the modular addition with a constant $\oplus c$ was proposed in [Azi+20]. For the round function of the block cipher Simon [Bea+15], $f_{a,b,c}(x) = (x \ll a) \land (x \ll b) \land (x \ll c)$, an XOR differential model was obtained in [KLT15].

- **RX differential models.** Given a $\oplus$-linear bit-vector function $f$ that commutes with $\ll 1$, 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 $(\alpha, \beta)$ 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 $(\alpha, \beta)$. 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 $\alpha$ or $\beta$ 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 $(\alpha, \beta)$ is considered invalid if the number of non-zero words in $\alpha$ and $\beta$ 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 $f = f_r \circ f_{r-1} \circ \cdots \circ f_0$ is valid if the propagation probability of $\Gamma$ is non-zero, and in this case we define the propagation weight of $\Gamma$ as

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

A (bit-vector) characteristic model of $f = f_r \circ f_{r-1} \circ \cdots \circ f_0$ is a set of bit-vector constraints that models the propagation weight of characteristics over $f = f_r \circ f_{r-1} \circ \cdots \circ f_0$. 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 $\text{VC}_i$, $\text{POC}_i$ and $\text{WC}_i$ denote the validity, probability-one constraint and weight constraint, respectively, of the property model of $f_i$. Then, the constraints $\text{VC}$, $\text{POC}$ and $\text{WC}$ of the characteristic model are given by

$$\text{VC}(\gamma_0, \ldots, \gamma_r) = \text{VC}_0(\gamma_0, \gamma_1) \land \cdots \land \text{VC}_{r-1}(\gamma_{r-1}, \gamma_r)$$

$$\text{POC}(\gamma_0, \ldots, \gamma_r) = \text{POC}_0(\gamma_0, \gamma_1) \land \cdots \land \text{POC}_{r-1}(\gamma_{r-1}, \gamma_r)$$

$$\text{WC}(w, \gamma_0, \ldots, \gamma_r) = \exists w_0, \ldots, w_{r-1} : (w = w_0 \oplus \cdots \oplus w_{r-1}) \land$$

$$\text{WC}_0(w_0, \gamma_0, \gamma_1) \land \cdots \land \text{WC}_{r-1}(w_r, \gamma_{r-1}, \gamma_r)$$

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 $\text{WC}$, where $\ell_i$ is the number of fractional bits and $\epsilon_i$ the error bound of $\text{WC}_i$. Note also that the bitsize of the weight variables in $\text{WC}$ might need to be increased (by left concatenating with zeros) to avoid overflows in $w = w_0 \oplus \cdots \oplus w_{r-1}$.

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 on 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 \((VC, POC, WC)\) 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

\[
\exists \gamma_0, \gamma_1, \ldots, \gamma_r, w, w': \\
VC(\gamma_0, \gamma_1, \ldots, \gamma_r) \\
WC(w', \gamma_0, \gamma_1, \ldots, \gamma_r) \\
w = \text{Truncate}(w', \ell)
\]

where \(\text{Truncate}(w', \ell)\) ignores the \(\ell\) 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 \(\epsilon\) 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 
\[
(\gamma_0 \neq \gamma_0') \lor (\gamma_1 \neq \gamma_1') \lor \cdots \lor (\gamma_r \neq \gamma_r'),
\]
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 \( \Gamma_0 \) over the simple function \( f_0 \) and then using the integer weight of \( \Gamma_0 \) 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 \((\Gamma_{KS}, \Gamma_{Ek})\), and constraining the sum of the propagation weight of \( \Gamma_{KS} \) and the propagation weight of \( \Gamma_{Ek} \) 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-scheduling zero weight faster.

For some properties such as the difference properties, the propagation probability of a global property \((\alpha, \beta)\) can be estimated by summing the propagation probabilities of all characteristics with input property \( \alpha \) and output property \( \beta \). In this case, the probability of \((\alpha, \beta)\) 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 \((\alpha, \beta)\) be a property of \( f \) for some bit-vector constants \( \alpha \) and \( \beta \). The brute-force and miss-in-the-middle methods are based on the fact that if the bit-vector SMT problem

\[
\exists \gamma_0, \gamma_1, \ldots, \gamma_r : VC(\gamma_0, \gamma_1, \ldots, \gamma_r) \land (\alpha = \gamma_0) \land (\beta = \gamma_r)
\]

(6)
is unsatisfiable, then \((\alpha, \beta)\) has zero propagation probability. The difference between these two methods is the choice of the properties \((\alpha, \beta)\). 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 \(\Gamma_0\) and \(\Gamma_2\), where the characteristic \(\Gamma_0\) (resp. \(\Gamma_2\)) covers the first (resp. second) half of cipher, such that the output difference \(\alpha\) of \(\Gamma_0\) does not match the input difference \(\beta\) of \(\Gamma_2\) 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 \(VC_i(\gamma_i, \gamma_{i+1})\) and \(POC_i(\gamma_i, \gamma_{i+1})\) be the validity and probability-one constraints, respectively, of the characteristic model of \(f_i\). First, a pair of probability-one characteristics \((\Gamma_0, \Gamma_2)\) of \(f_0\) and \(f_2\) is found by solving the bit-vector SMT problem

\[
\exists \gamma_0, \gamma_1, \gamma_2, \gamma_3 : POC_0(\gamma_0, \gamma_1) \land POC_2(\gamma_2, \gamma_3). \tag{7}
\]

Let \((\alpha', \alpha)\) be the input and output properties of \(\Gamma_0\) and \((\beta, \beta')\) the input and output properties of \(\Gamma_2\). Then, a similar problem than Eqn. (6) is built for the property \((\alpha, \beta)\) of \(f_1\). If the problem is unsatisfiable, \((\alpha, \beta)\) is a zero-probability property of \(f_1\), and by construction \((\alpha', \beta')\) 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 \((\alpha, \beta)\) 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

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

If satisfiable, a solution of this problem is an assignment of the variables \((\gamma_0, \gamma_r)\) 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 \((\gamma_0, \gamma_r)\) 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 $(\alpha, \beta)$ 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

\(^3\) 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 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.

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.](image)

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 schedules 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-scheduling 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. 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-scheduling 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 \( (\gamma_0, \gamma_{r-1}) \) computed by evaluating \( f \) for many inputs satisfying \( \gamma_0 \). 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

\footnote{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.}
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 $\boxplus_c(x) = x \boxplus c$ and $\boxminus_c(x) = x \boxminus c$, the binary operators $\land, \lor, \boxplus, \boxminus$, 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 ($\land, \lor$, 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 $\lll_c(x) = x \lll c$ and $\ggg_c(x) = x \ggg c$, the binary operators $\land, \lor, \boxplus, \boxminus$, and the ternary operators bit-wise majority, bit-wise conditional and the Simon round function $f_{a,b,c}$. The models of $\lll_c$ and $\ggg_c$ 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 $\land$ and $\lor$ 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 $\alpha$ can propagate through a branch $x \mapsto (x, x)$ to multiple output masks (i.e., to any output mask $(\beta_0, \beta_1)$ 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 $x \mapsto x \oplus c$ or a differential model of the rotation by a variable $(x, y) \mapsto x \ll y,$
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


[KPR20] Liliya Kralova, Raluca Posteauca and Vincent Rijmen. ‘Cryptanalysis of the Permutation Based Algorithm SpoC’. In: INDOCRYPT.


