A Bit-Vector Differential Model for the Modular Addition by a Constant

Seyyed Arash Azimi, Adrián Ranea, Mahmoud Salmasizadeh,
Javad Mohajeri, Mohammad Reza Aref, and Vincent Rijmen

AsiaCrypt'20 December 8, 2020

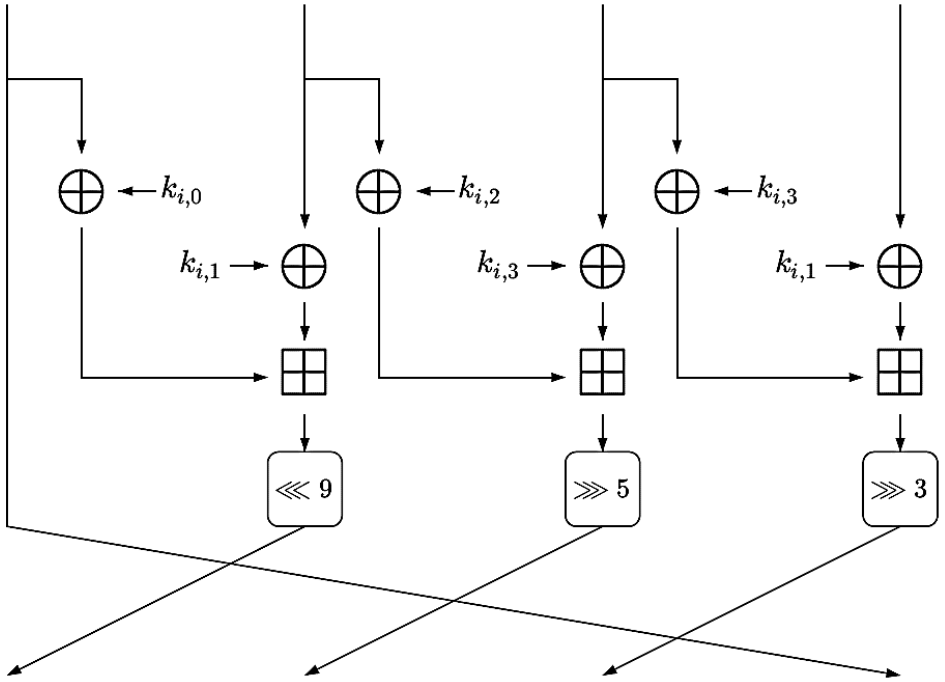
ARX Block Ciphers

ARX:

Addition: $\boxplus$

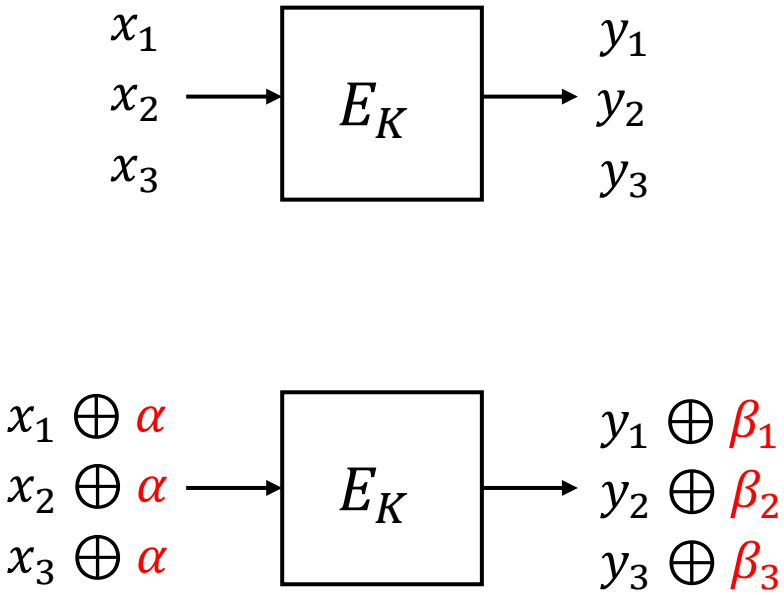
Rotation: $\lll$ $\ggg$

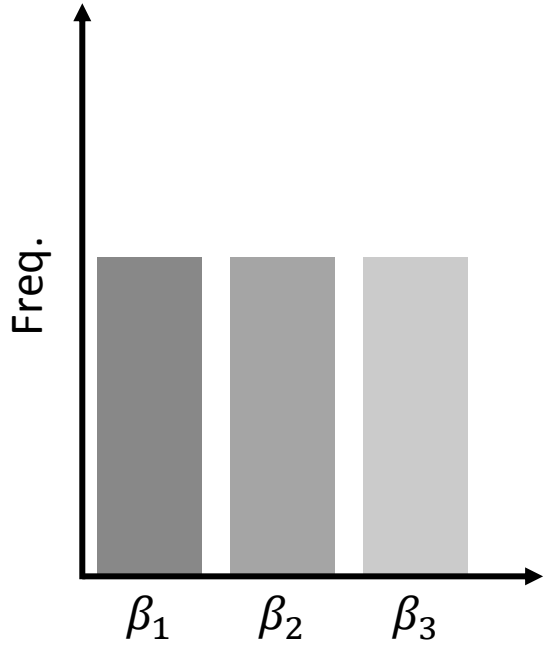
XOR: $\oplus$



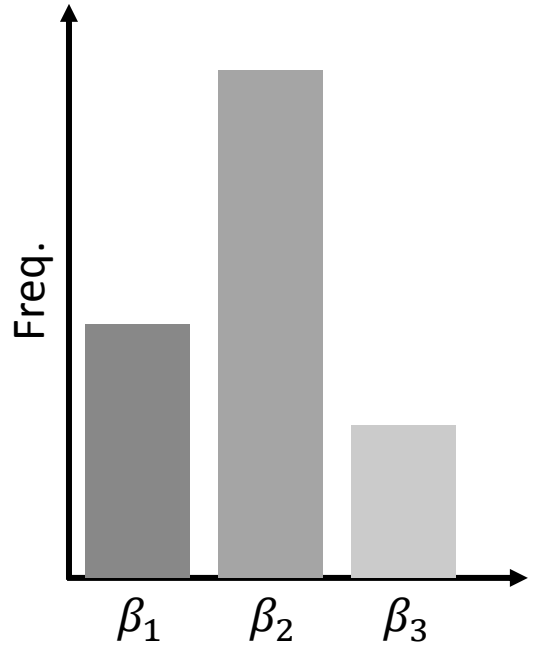
Examples: TEA, XTEA, HIGHT, LEA, SPECK, CHAM, etc.

Differential Cryptanalysis





Random Behavior



Non-Random Behavior

State-of-the-art ARX: SMT-based search for optimal characteristics.

Differential Models

SMT requires efficient bit-vector differential models.

Linear operator $\mathcal{L}$: bit-vector constraints representing $\alpha \xrightarrow{\mathcal{L}} \beta$.

- E.g.: $\mathcal{L}(x) = x \lll 1$, the constraint is $\beta = \alpha \lll 1$.

Non-Linear operator F:

- valid(α, β): True iff. $\Pr\left(\alpha \xrightarrow{F} \beta\right) \neq 0$;
- weight(α, β, w): True iff. $w = -\log_2 \Pr\left(\alpha \xrightarrow{F} \beta\right)$.

Differential Model for addition

Differential Model for n-bit modular addition $f(x_1, x_2) = x_1 \boxplus x_2$:

- valid$(\alpha_1, \alpha_2, \beta)$: $0 = eq(\overleftarrow{\alpha_1}, \overleftarrow{\alpha_2}, \overleftarrow{\beta}) \wedge (\alpha_1 \oplus \alpha_2 \oplus \beta \oplus \overleftarrow{\alpha_2})$;
- weight(α, β, w): $w = \mathrm{HW}(\neg eq(\alpha_1, \alpha_2, \beta) \ll 1)$.

Bit-vector complexity (formula size): $\mathcal{O}(\log n)$.

No differential model for constant addition $f_c(x) = x \boxplus c = x \boxplus_c$.

Modular addition VS Constant addition

Why not model $(\alpha \xrightarrow{\boxplus_c} \beta)$ as $(\alpha, 0 \xrightarrow{\boxplus} \beta)$?

$F, G: \mathbb{F}_2^n \to \mathbb{F}_2^m$ are <u>CCZ-equivalent</u> if: ($\mathcal{L}: \mathbb{F}_2^{n+m} \to \mathbb{F}_2^{n+m}$ affine perm.)

$$\Gamma_G = \{(x, G(x)), \forall x \in \mathbb{F}_2^n\} = \mathcal{L}(\{(x, F(x)), \forall x \in \mathbb{F}_2^n\}) = \mathcal{L}(\Gamma_F).$$

- $\boxplus$ is CCZ-equivalent to a quadratic function, $\boxplus_c$ not.
- Empirically, $\Pr(\alpha \xrightarrow{\boxplus_c} \beta) \neq \Pr(\alpha, 0 \xrightarrow{\boxplus} \beta)$.

Modeling $(\alpha \xrightarrow{\boxplus_c} \beta)$ is much harder than $(\alpha_1, \alpha_2 \xrightarrow{\boxplus} \beta)$.
- DDT entries of $\boxplus$ are power of 2, not the case for $\boxplus_c$.

Machado's Algorithm

Input: (α, β)

Output: $\Pr(\alpha \xrightarrow{\boxplus_c} \beta)$

$p, d \leftarrow 1, 0$

for $i = 0, \dots, n-1$

if $(\alpha_i, \beta_i, \alpha_{i+1} \oplus \beta_{i+1}) = (0,0,0)$

$d \leftarrow (c_i + d)/2$

elif $(\alpha_i, \beta_i, \alpha_{i+1} \oplus \beta_{i+1}) = (0,0,1)$

$\vdots$

elif $(\alpha_i, \beta_i, \alpha_{i+1} \oplus \beta_{i+1}) = (1,1,1)$

$p \leftarrow p \times (c_i + d - 2c_i \times d)$

$d \leftarrow 1/2$

Return p

Not suitable for automated models:

- Loops → BV complexity $\geq \mathcal{O}(n)$
- Floating-point arithmetic
- No direct translation to weight

Obtaining Bit-Vector Differential Model of $\boxplus_c$

valid(α, β)

- Verify some consecutive equations with a single Carry function.

weight(α, β, w)

- Known $\mathcal{O}(\log n)$ function: HW, reverse, leading zeros
- Bit-vector approximation of the binary logarithm.

  Example: $x = (000111001)_2 \quad \Rightarrow \quad \log_2 x \approx 5 + (0.1100)_2$

- New $\mathcal{O}(\log n)$ functions for parallel logarithm.

  Example: $x = (*\,100\,*\,0101\,*)_2$

  $m = (0110\ 0\ 1110\ 0)_2$

  $APL(x, m) = (\lfloor \log_2(100)_2 \rfloor + (0.00)_2) + (\lfloor \log_2(0101)_2 \rfloor + (00.01)_2)$

Our Differential Model

Input: (α, β)

Output: valid$(\alpha \xrightarrow{\boxplus_c} \beta)$

$s_{00*} \leftarrow \neg(\alpha \ll 1) \wedge \neg(\beta \ll 1)$

$s_{11*} \leftarrow (\alpha \ll 1) \wedge (\beta \ll 1)$

$s_{**1} \leftarrow \alpha \oplus \beta$

$t \leftarrow \neg(s_{00*} \ll 1)$

$b \leftarrow \mathrm{Carry}(s_{00*} \wedge \neg c', t)$

$g \leftarrow (s_{**1} \oplus c') \wedge (b \vee t)$

Return $0 = (s_{00*} \wedge s_{**1}) \vee (s_{11*} \wedge g)$

Bit-Vector complexities:

- valid: $\mathcal{O}(1)$
- weight: $\mathcal{O}(\log n)$

Input: (α, β, w)

Output: weight$(\alpha \xrightarrow{\boxplus_c} \beta, w)$

$s_{000} \leftarrow \neg(\alpha \ll 1) \wedge \neg(\beta \ll 1)$

$s'_{000} \leftarrow s_{000} \wedge \neg\mathrm{LZ}(\neg s_{000})$

$t \leftarrow \neg s'_{000} \wedge (s'_{000} \gg 1)$

$t' \leftarrow s'_{000} \wedge \left(\neg(s'_{000} \gg 1)\right)$

$s \leftarrow \left((c \ll 1) \wedge t\right) \boxplus \left(c \wedge (s'_{000} \gg 1)\right)$

$q \leftarrow \left(\left(\neg\left((c \ll 1) \oplus \alpha \oplus \beta\right)\right) \gg 1\right) \wedge t'$

$d \leftarrow \mathrm{RevCarry}(s'_{000}, q) \vee q$

$b \leftarrow \left(q \boxminus (s \wedge d)\right) \vee (s \wedge \neg d)$

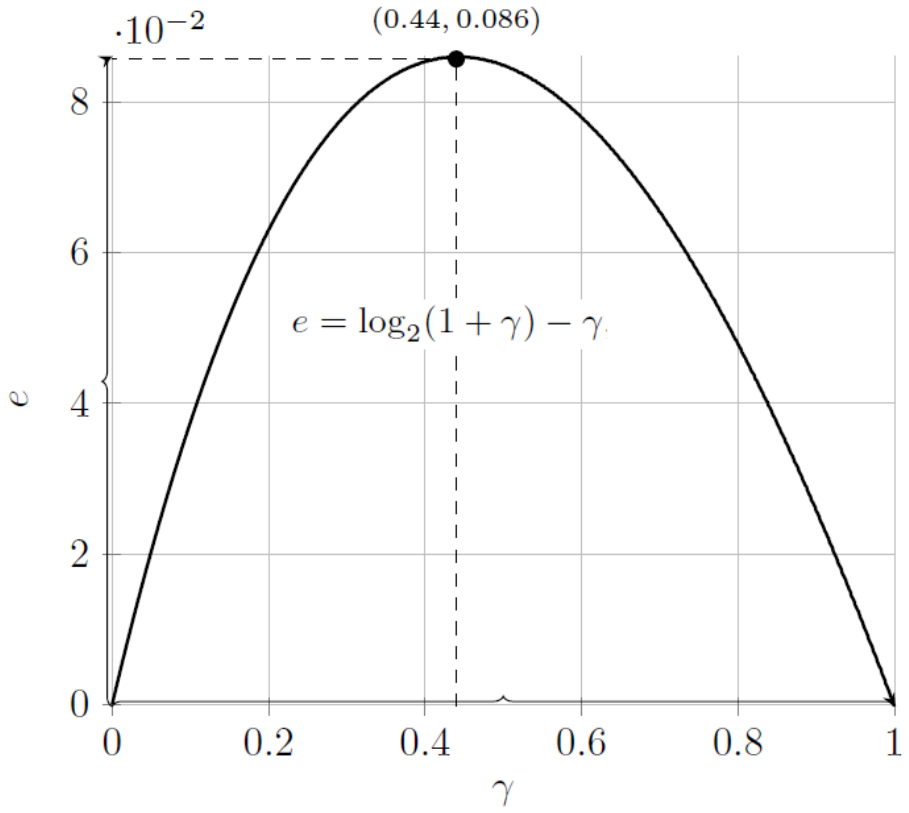
$x \leftarrow (b \wedge s'_{000}), s'_{000} \ll 1$

$i \leftarrow \mathrm{HW}\left((\alpha \oplus \beta) \ll 1\right) \boxplus \mathrm{HW}(s'_{000}) \boxminus \mathrm{PL}(x)$

$f \leftarrow \mathrm{PT}\left(b \ll 1, \mathrm{RevCarry}(x)\right)$

Return $w = (i \ll 4) \boxminus f$

Error Analysis



Error of approx. $\log_2$ w/o truncation.

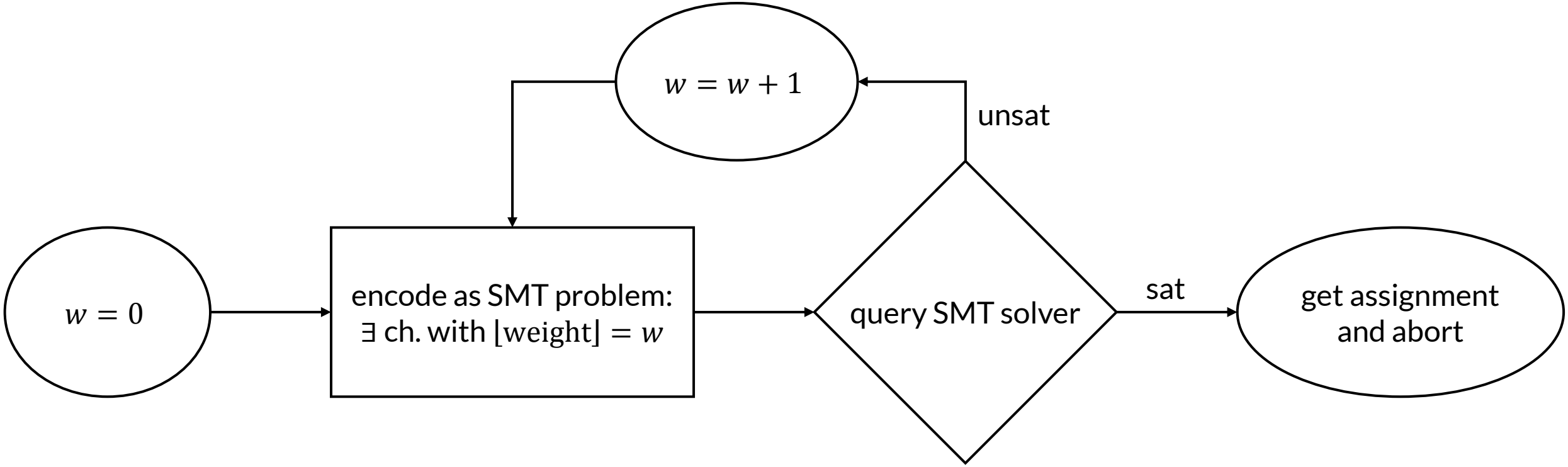
Truncating:

- Same error bounds for 4 or more fraction bits.

Theorem:

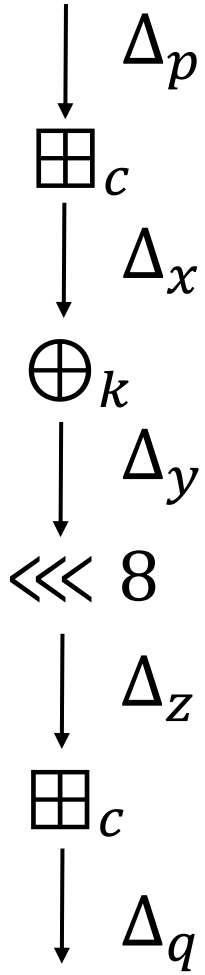
The weight approx. error for 4 fraction bits is bounded by $[-0.029n, 0]$.

SMT-based Search of Characteristic



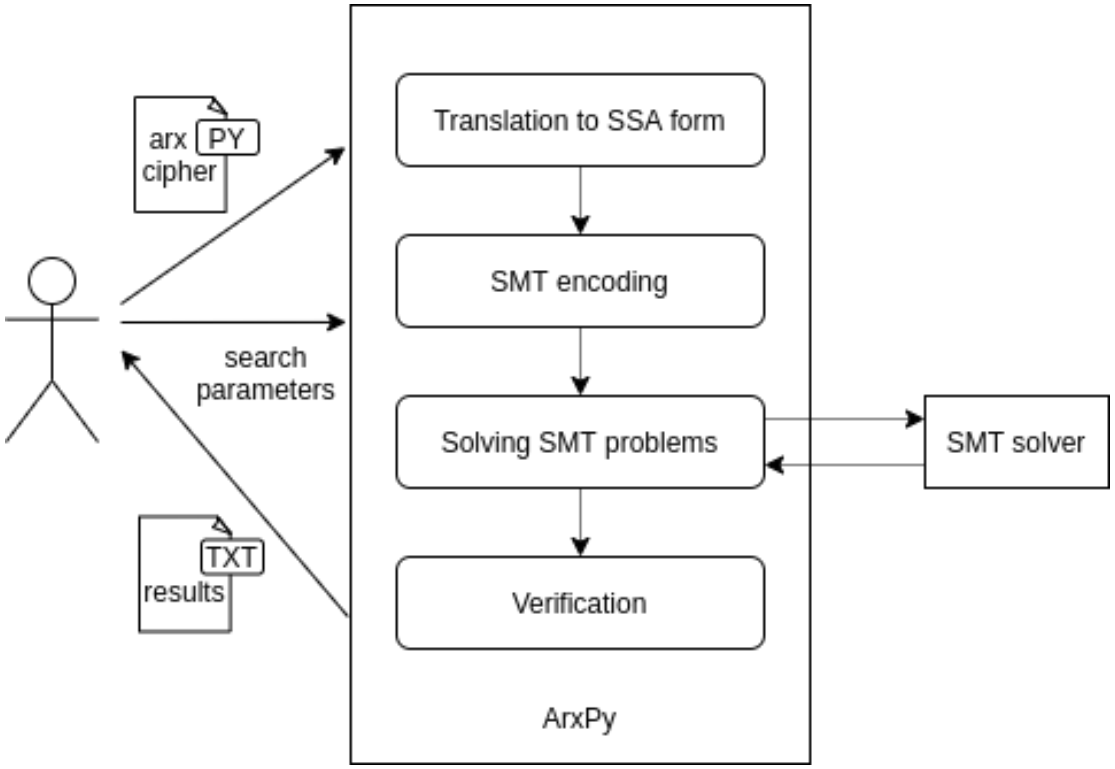
- speed up by searching iteratively over the rounds
- optimal characteristic found under standard assumptions

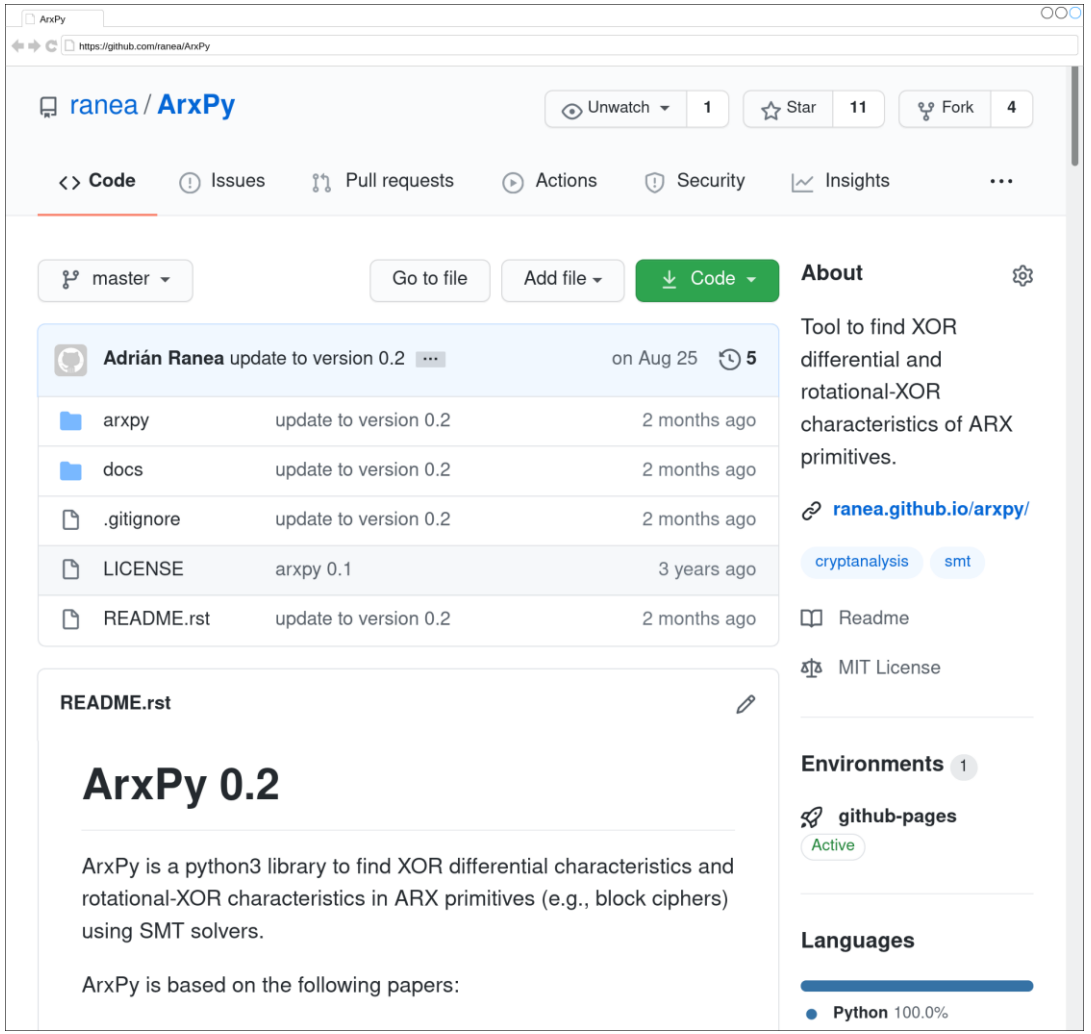
Encoding the SMT Problems



$$
\begin{aligned}
\exists\, \Delta_p, \Delta_x, \Delta_y, \Delta_z, \Delta_q, w_1, w_2 :\\
&\mathrm{valid}_{\boxplus_c}(\Delta_p, \Delta_x)\,,\\
&\mathrm{weight}_{\boxplus_c}(\Delta_p, \Delta_x, w_1)\,,\\
&\Delta_y = \Delta_x \oplus 0\,,\\
&\Delta_z = \Delta_y \lll 8\,,\\
&\mathrm{valid}_{\boxplus_c}(\Delta_z, \Delta_q)\,,\\
&\mathrm{weight}_{\boxplus_c}(\Delta_z, \Delta_q, w_2)\,,\\
&\text{target weight} = w_1 \boxplus w_2
\end{aligned}
$$

ArxPy


arx PY
cipher
search
parameters
TXT
results
Translation to SSA form
SMT encoding
Solving SMT problems
Verification
ArxPy
SMT solver


ranea / ArxPy
Unwatch 1
Star 11
Fork 4
Code
Issues
Pull requests
Actions
Security
Insights
master
Go to file
Add file
Code
Adrián Ranea update to version 0.2
on Aug 25
5
arxpy update to version 0.2 2 months ago
docs update to version 0.2 2 months ago
.gitignore update to version 0.2 2 months ago
LICENSE arxpy 0.1 3 years ago
README.rst update to version 0.2 2 months ago
README.rst
ArxPy 0.2
ArxPy is a python3 library to find XOR differential characteristics and rotational-XOR characteristics in ARX primitives (e.g., block ciphers) using SMT solvers.
ArxPy is based on the following papers:
About
Tool to find XOR differential and rotational-XOR characteristics of ARX primitives.
ranea.github.io/arxpy/
cryptanalysis
smt
Readme
MIT License
Environments 1
github-pages
Active
Languages
Python 100.0%

Experiments

- Constant addition mostly used in the key schedule.
- We searched for related-key ch. of TEA, XTEA, HIGHT and LEA.
- Standard assumptions do not hold, we verify each ch. empirically
  - We split each rk ch. found in smaller ch.
  - Check each small ch. with 2^{20} pairs for 2^{10} keys.
  - If a small ch. has zero probability for all keys, full ch. discarded.
- Results obtained with ArxPy + Boolector running for 1 week.

Cipher	Rounds	$(w_{KS}, \overline{w_{KS}})$	$(w_E, \overline{w_E})$	% valid keys	Reference
TEA	full	0	0	-	Previous work
	full	(0, 0)	(0, 0)	100 %	This paper
XTEA	16	0	32	-	Previous work
	18	(0, 0)	(57, 49)	48 %	This paper
	18	17	19	-	Previous work
	18	(4, 3)	(16, 14)	100 %	This paper
	27	(6, 5)	(40, 39)	7 %	This paper
LEA	11	high	<128	-	Previous work
	7	(2, 4)	(36, 34)	100 %	This paper
HIGHT	10	0	12	-	Previous work
	15	(0, 0)	(45, 42)	8 %	This paper
	12	2	19	-	Previous work
	14	(13, 9)	(14, 11)	17 %	This paper