AutoDiVer

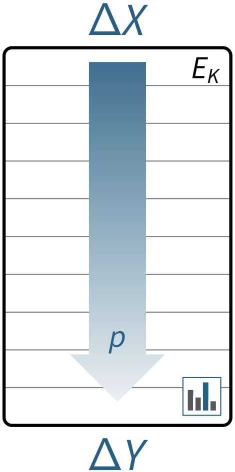
Automatic Verification of Differential Characteristics

Marcel Nageler, Shibam Ghosh, Marlene Jüttler, Maria Eichlseder

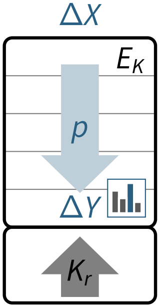
FSE 2025 – Rome

Differential Characteristics

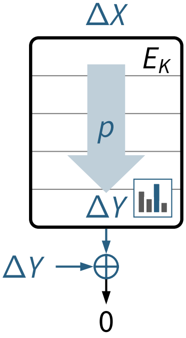
Method



Attack Goals



key recovery



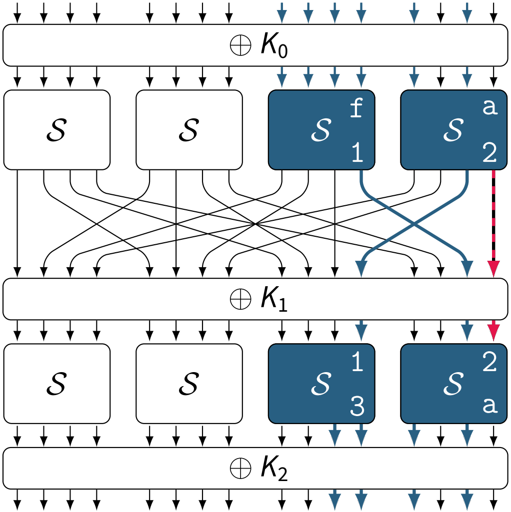
collision,
forgery

...

Open Questions

- Is the differential characteristic **possible**?
- What is the **expected differential probability** of the characteristic?
- Does the probability **depend on the key**?
- Does the probability of the characteristic accurately predict the differential probability?

Key Dependency in a Differential Characteristic



$\mathcal{YDDT}(\mathrm{a}, 2) = \{0, 2\}$

$K_{1,0}$ must be 0

$\mathcal{XDDT}(2, \mathrm{a}) = \{0, 2\}$

Potential Remedies

- Experimental verification
  - Infeasible for low-probability characteristics
- Quasidifferential Trails [BR22]
  - Theoretical Framework
  - Uses linear masks instead of values
- Propagating constraints [PT22]
  - Each active S-box forms a half-constraint
  - Connect two half-constraints to find a dependency
- SAT-based verification
  - Create cipher model in SAT
  - Recover more information about characteristic

AutoDiVer

Input

- Cipher description as CNF (already available for many ciphers)
- Differential Characteristic

Output

- Necessary **linear and nonlinear conditions** for keys
  - Identify **relevant S-boxes**
- **Estimated probability** averaged over all keys / specific key
- Estimate **number of valid keys**

https://github.com/isec-tugraz/AutoDiVer

Creating the Cipher Description

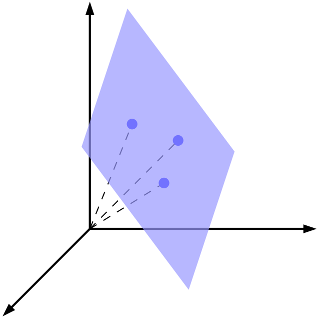
- Encode block cipher as a CNF
  - Linear layer
  - S-boxes
  - Key schedule with round constants
- Active S-boxes need additional constraints
  - Must follow solution set $\{x, y : \mathcal{S}(x) = y \wedge \mathcal{S}(x \oplus \Delta_\mathrm{i}) = y \oplus \Delta_\mathrm{o}\}$
  - CNF created automatically with `espresso`

Applying a SAT Solver

- Use a SAT solver to verify whether any solution exists
  - can detect **impossible** characteristics
  - or **proves** characteristic is **possible**
- Solve with a SAT solver for many **random keys**
  - approximate the number of valid keys
  - SAT Solver learns a **clause for each UNSAT key**

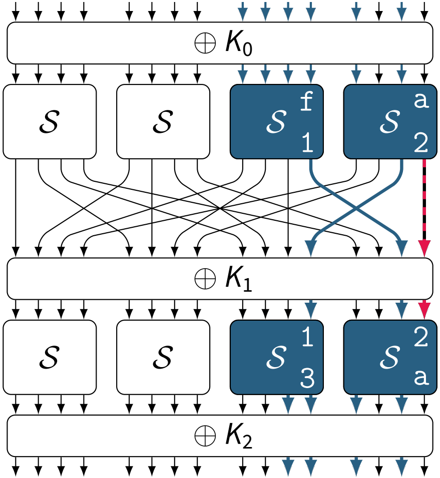
Systematically Learning Linear Conditions

- Solve SAT problem $\mathcal{F}$ with random seed $k+1$ times.
- Calculate affine hull of the $k+1$ keys.
  - Formulate conjecture as $A \cdot K = c$.
- Solve SAT problem $\mathcal{F} \wedge A \cdot K \neq c$.
  - UNSAT $\rightarrow$ proof that $A \cdot K = c$.
  - SAT $\rightarrow$ repeat previous step.



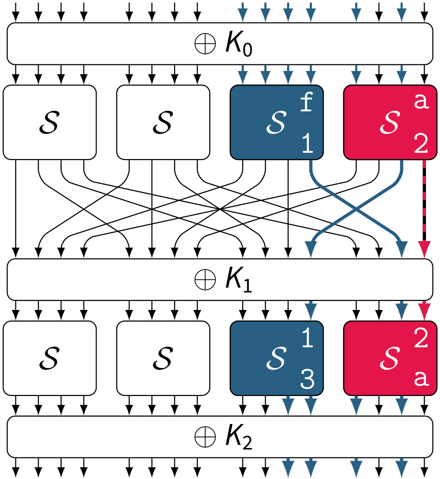
Finding Affine Hull: Example

- Solve $\mathcal{F}$ with random seed 49 times
  - 49 values for (K_0, K_1, K_2) that span an affine space of dimension 47
- Conjecture based on affine space: $K_{1,0} = 0$
- Solve $\mathcal{F} \wedge K_{1,0} \neq 0$
  - UNSAT $\rightarrow K_{1,0}$ must be 0



Finding Relevant S-box: Example

- Add assumption variables $s_{r,i}$ for each S-box
  - $s_{r,i}$ implies clauses for S-box i of round r
- Solve $\mathcal{F} \wedge K_{1,0} \neq 0$ under assumption of $s_{r,i}$
  - UNSAT, conflict: $\neg s_{0,0} \vee \neg s_{1,0}$



Background: Approximate Model Counting

- Research field related to SAT solving
  - We use ApproxMC [SGM20]
- Given CNF formula F, tolerance ε, and confidence δ
  - Optional: projection set of variables to count over
- output **approximate solution count** c
  - $\frac{1}{1+\varepsilon} \leq \frac{c}{|\mathrm{sol}(F)|} \leq 1 + \varepsilon$
  - with probability $p \geq 1 - \delta$
- use this to count probability / number of valid keys

Counting number of valid keys

- The model counter provides one extra input:
  - the sampling set

  $\rightarrow$ count how many assignments for this subset of variables exist
- use this to count key space

Results

- Implementation for many ciphers
  - strongly aligned SPN: **SKINNY, Midori**
  - pure SPN: **GIFT, PRESENT, RECTANGLE**
  - weakly aligned SPN: **Ascon, SPEEDY, Pyjamask**
  - Feistel: **WARP**
  - ARX: **SPECK**
- Many key dependencies confirmed and some new ones
- Observations
  - Finding affine hull is most effective for more rounds
  - ApproxMC reliable but slow for many rounds

Example Results

For Midori64 characteristic with $p = 2^{-52}$ ([Zha+20, Tab. 5])

- Deterministic results
  - Affine Hull: 17 linear conditions (1.4 seconds)
  - No extra nonlinear conditions found ($n = 100$k, 35 seconds)
  - key set size $\leq 2^{111}$
- Statistical results
  - $p = 2^{-52 \pm 0.14}$ with probability 0.99 (95 seconds, $\varepsilon = 0.1$, $\delta = 0.01$)
    - (averaged over all 2^{128} keys)
  - key set size $= 2^{111 \pm 0.14}$ with probability 0.99 (57 seconds, $\varepsilon = 0.1$, $\delta = 0.01$)

Example Usage

Terminal

```
❯ autodiver midori64 trails/midori64/midori64_zhww_r5_1.npz affine-hull
INFO version: 0.5, git_commit: 1a68e0b77eec7183b87374a92f1ff5c030a4f964, git_changed_files: []
INFO loaded characteristic with 5 rounds from trails/midori64/midori64_zhww_r5_1.npz with ddt probability 2**-52.0
INFO generated XorCNF over 1232 variables with 1396 clauses and 640 xor clauses
gathering valid keys: 100%|████████████████████████████████████████| 129/129 [00:00<00:00, 219.05it/s]
INFO gathered keys span affine space of dimension 111
INFO solving for counterexample
INFO RESULT no counterexample found -> conditions on key are necessary
INFO RESULT key[0, 0, 0, 3] ⊕ key[0, 1, 0, 3] = 1
INFO RESULT key[0, 0, 3, 3] ⊕ key[0, 1, 3, 3] ⊕ key[0, 3, 3, 3] = 0
INFO RESULT key[0, 1, 1, 3] ⊕ key[0, 3, 1, 3] = 0
INFO RESULT key[0, 1, 2, 3] ⊕ key[0, 3, 2, 3] = 1
INFO RESULT key[0, 3, 0, 3] = 0
INFO RESULT key[1, 0, 1, 2] ⊕ key[1, 1, 1, 2] ⊕ key[1, 3, 1, 2] = 0
INFO RESULT key[1, 0, 1, 3] ⊕ key[1, 1, 1, 3] ⊕ key[1, 3, 1, 3] = 0
INFO RESULT key[1, 0, 3, 2] ⊕ key[1, 1, 3, 2] = 0
INFO RESULT key[1, 0, 3, 3] ⊕ key[1, 1, 3, 3] = 0
INFO RESULT key[1, 1, 0, 0] = 1
INFO RESULT key[1, 1, 0, 3] = 0
INFO RESULT key[1, 2, 0, 0] ⊕ key[1, 3, 0, 0] = 0
INFO RESULT key[1, 2, 0, 3] ⊕ key[1, 3, 0, 3] = 0
INFO RESULT key[1, 2, 1, 2] ⊕ key[1, 3, 1, 2] = 0
INFO RESULT key[1, 2, 1, 3] ⊕ key[1, 3, 1, 3] = 0
INFO RESULT key[1, 3, 2, 2] = 0
INFO RESULT key[1, 3, 2, 3] = 0
❯
```

Conclusion

- Tool for verification of differential characteristics
- We show whether a characteristic is **possible** or not
- We estimate the expected **probability** of a characteristic
- We find linear and nonlinear **conditions** on the key
- Probability of the **differential** might be larger than of characteristic
- **Variation of probability** across valid keys needs different tools

Bibliography I

[BR22] Tim Beyne and Vincent Rijmen. **Differential Cryptanalysis in the Fixed-Key Model.** CRYPTO 2022. Vol. 13509. LNCS. Springer, 2022, pp. 687–716. DOI: 10.1007/978-3-031-15982-4_23.

[PT22] Thomas Peyrin and Quan Quan Tan. **Mind Your Path: On (Key) Dependencies in Differential Characteristics.** IACR Transactions on Symmetric Cryptology 2022.4 (2022), pp. 179–207. DOI: 10.46586/TOSC.V2022.I4.179-207.

[SGM20] Mate Soos, Stephan Gocht, and Kuldeep S. Meel. **Tinted, Detached, and Lazy CNF-XOR Solving and Its Applications to Counting and Sampling.** CAV 2020. Vol. 12224. LNCS. Springer, 2020, pp. 463–484. DOI: 10.1007/978-3-030-53288-8_22.

[Zha+20] Hongluan Zhao et al. **MILP-Based Differential Cryptanalysis on Round-Reduced Midori64.** IEEE Access 8 (2020), pp. 95888–95896. DOI: 10.1109/ACCESS.2020.2995795.