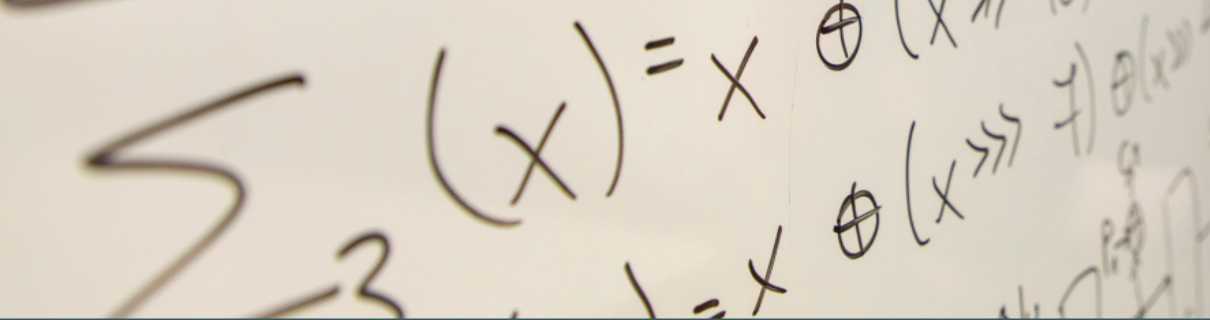


Automating the Search for Cryptanalytic Attacks

Maria Eichlseder

- FSE 2024
- Leuven · 26 Mar 2024

› https://iaik.tugraz.at

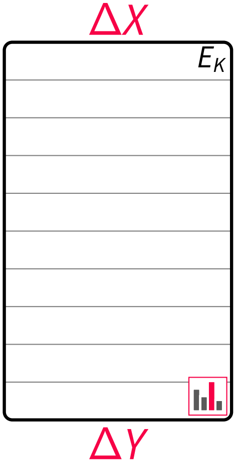
Outline

- Motivation
- Finding Distinguishers with MILP/SAT Solvers
  - Mixed-Integer Linear Programming (MILP)
  - Boolean Satisfiability and Constraint Programming (SAT/SMT, CP)
- Dedicated Algorithms
- Optimized Key Recovery Attacks
- Frameworks

Motivation

Differential Cryptanalysis [BS90]

Method



Attack Goals

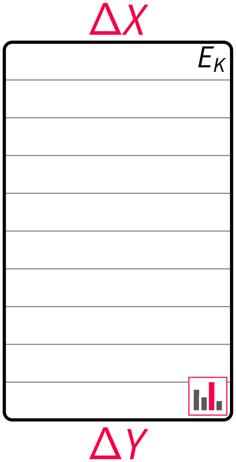
collision,
forgery

key recovery

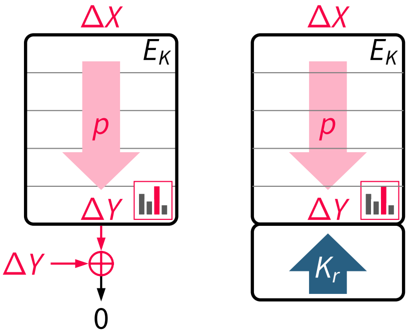
. . .

Differential Cryptanalysis [BS90]

Method



Attack Goals



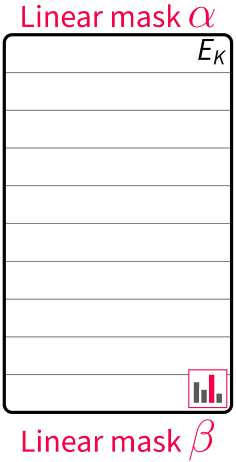
collision, forgery

key recovery

. . .

Linear Cryptanalysis [Mat93]

Method



Attack Goals

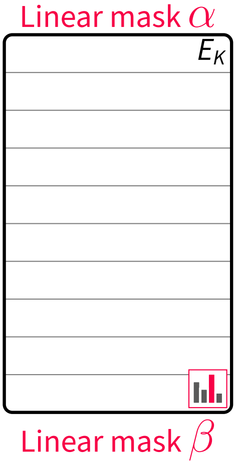
confidentiality

key recovery

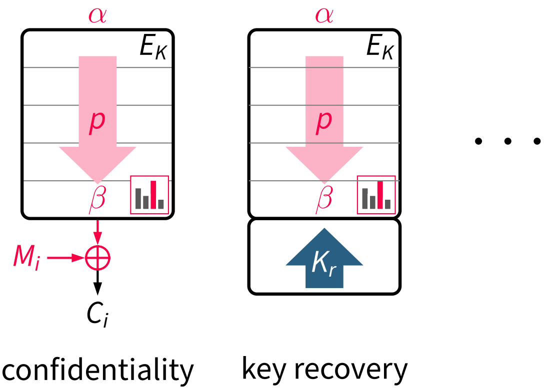
. . .

Linear Cryptanalysis [Mat93]

Method

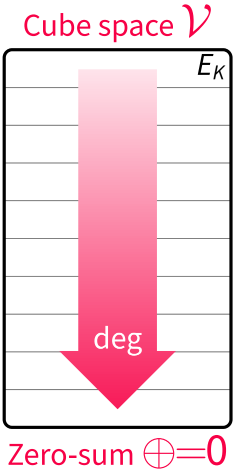


Attack Goals



Integral Cryptanalysis [Lai94; Knu94; KW02]

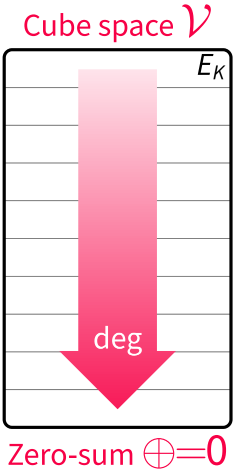
Method



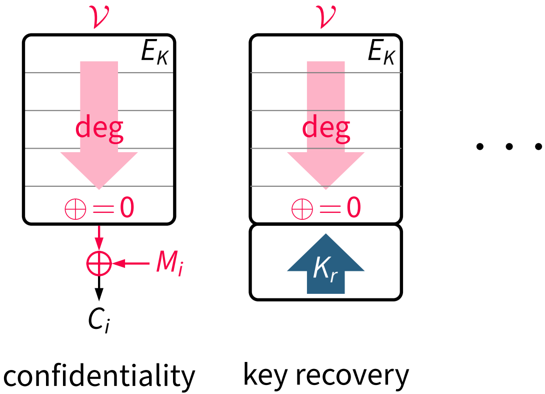
Attack Goals

Integral Cryptanalysis [Lai94; Knu94; KW02]

Method



Attack Goals



How to Find Distinguishers

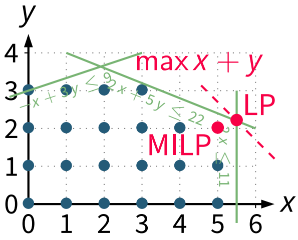
- By hand
- Dedicated solvers
- General-purpose solvers:
  - SAT/SMT (Boolean SATisfiability/Sat. Modulo Theories)
  - MILP (Mixed Integer Linear Programming)
  - CP (Constraint Programming)

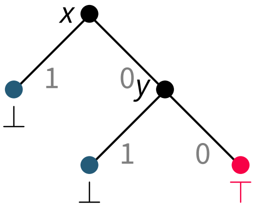
How to Find Distinguishers

- By hand
- Dedicated solvers
- General-purpose solvers:
  - SAT/SMT (Boolean SATisfiability/Sat. Modulo Theories)
  - MILP (Mixed Integer Linear Programming)
  - CP (Constraint Programming)

How to Find Distinguishers

- By hand
- Dedicated solvers
- General-purpose solvers:
  - SAT/SMT (Boolean **SAT**isfiability/**S**at. **M**odulo **T**heories)
  - MILP (**M**ixed **I**nteger **L**inear **P**rogramming)
  - CP (**C**onstraint **P**rogramming)





Finding Distinguishers with MILP/SAT Solvers

Basic Approach

- Model constraints that characterize correct characteristics/solutions
  - Coarse-grained: truncated patterns (which S-boxes are active?)
  - Fine-grained: precise differences/masks
- Model cost (if applicable)
- Express the search goal: any one / all / best / good solution(s)?

Mixed-Integer Linear Program (MILP)

Linear Programming (LP) is a method to solve optimization problems

- on the real-valued, positive decision variables $x \in \mathbb{R}^d, x \geq 0$
- with a linear objective function (min or max) $f(x) = c^\top x = \sum_{i=1}^{d} c_i x_i$
- under J linear constraints (s.t.) $Ax \leq b$, i.e., $\sum_{i=1}^{d} a_{ji} x_i \leq b_j$ for $1 \leq j \leq J$:

$$\max_{x \in \mathbb{R}^d} \{c^\top x \mid Ax \leq b \wedge x \geq 0\}$$

Mixed-Integer Linear Programming (MILP) allows some of the decision variables to be constrained to integer values: $x \in \mathbb{Z}^i \times \mathbb{R}^{d-i}$.

Mixed-Integer Linear Program (MILP)

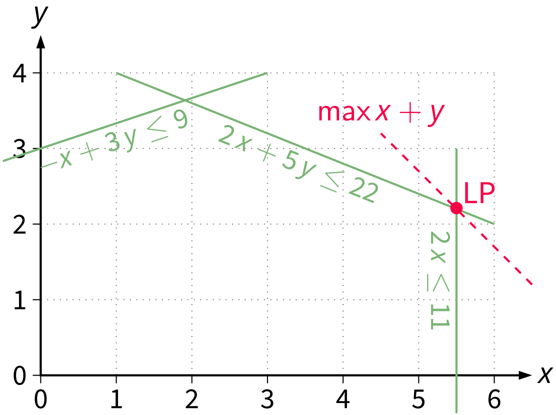
Linear Programming (LP) is a method to solve optimization problems

- on the real-valued, positive decision variables $x \in \mathbb{R}^d, x \geq 0$
- with a linear objective function (min or max) $f(x) = c^\intercal x = \sum_{i=1}^{d} c_i x_i$
- under J linear constraints (s.t.) $Ax \leq b$, i.e., $\sum_{i=1}^{d} a_{ji} x_i \leq b_j$ for $1 \leq j \leq J$:

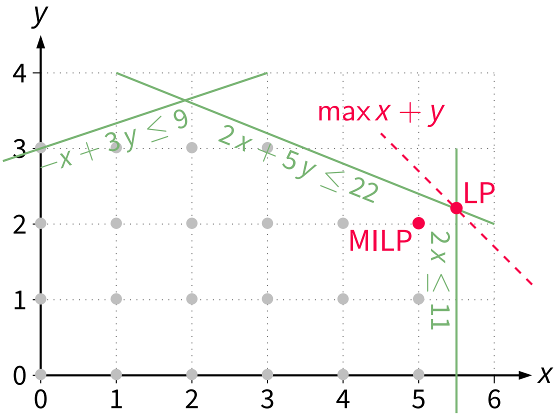
$$\max_{x \in \mathbb{R}^d} \{c^\intercal x \mid Ax \leq b \wedge x \geq 0\}$$

Mixed-Integer Linear Programming (MILP) allows some of the decision variables to be constrained to integer values: $x \in \mathbb{Z}^i \times \mathbb{R}^{d-i}$.

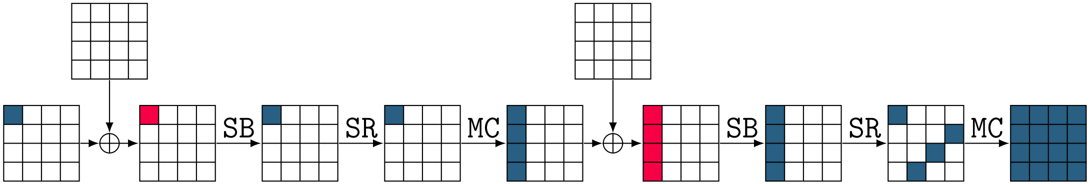
LP vs. MILP



LP vs. MILP



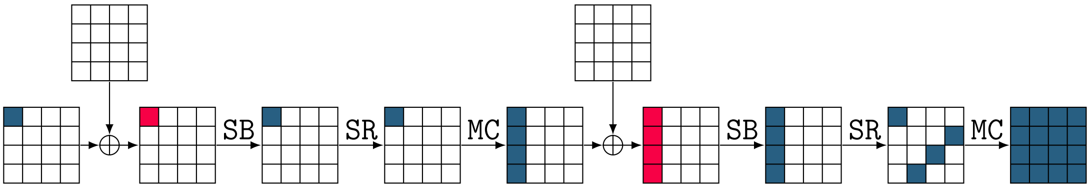
MILP – Example application: AES [MWGP11; WW11]



Variables: 1 binary variable per state byte (active/inactive)

- AddRoundKey: input = output
- SubBytes: input = output, cost = sum(inputs)
- ShiftRows: variable renaming
- MixColumns: for each active column: sum(inputs) + sum(outputs) $\geq$ 5 (= $\mathcal{B}$)

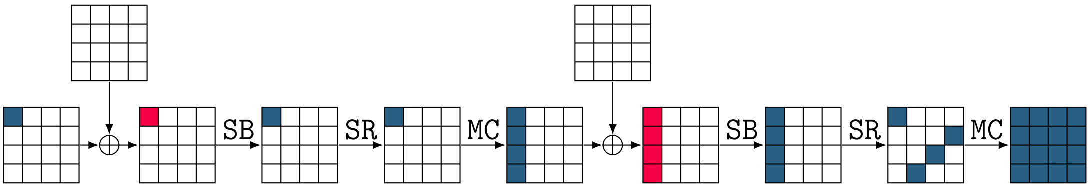
MILP – Example application: AES [MWGP11; WW11]



Variables: 1 binary variable per state byte (active/inactive)

- AddRoundKey: input = output
- SubBytes: input = output, cost = sum(inputs)
- ShiftRows: variable renaming
- MixColumns: for each active column: sum(inputs) + sum(outputs) ≥ 5 $(= \mathcal{B})$

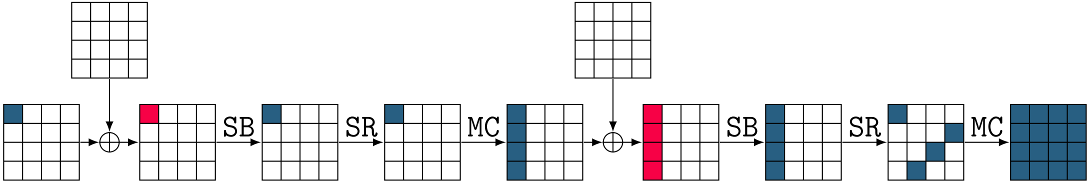
MILP – Example application: AES [MWGP11; WW11]



Variables: 1 binary variable per state byte (active/inactive)

- AddRoundKey: input = output
- SubBytes: input = output, cost = sum(inputs)
- ShiftRows: variable renaming
- MixColumns: for each active column: sum(inputs) + sum(outputs) ≥ 5 ($= \mathcal{B}$)

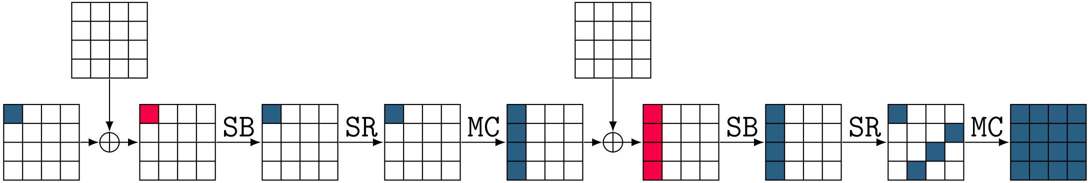
MILP – Example application: AES [MWGP11; WW11]



Variables: 1 binary variable per state byte (active/inactive)

- AddRoundKey: input = output
- SubBytes: input = output, cost = sum(inputs)
- ShiftRows: variable renaming
- MixColumns: for each active column: sum(inputs) + sum(outputs) ≥ 5 $(= \mathcal{B})$

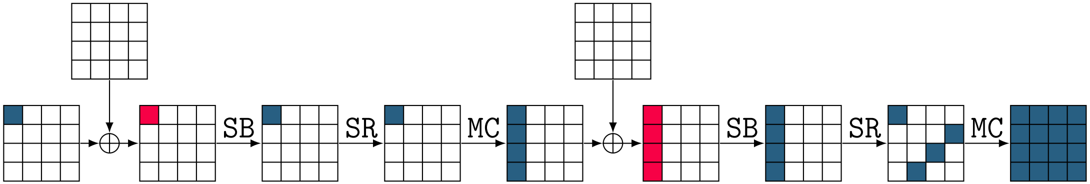
MILP – Example application: AES [MWGP11; WW11]



Variables: 1 binary variable per state byte (active/inactive)

- AddRoundKey: input = output
- SubBytes: input = output, cost = sum(inputs)
- ShiftRows: variable renaming
- MixColumns: for each active column: sum(inputs) + sum(outputs) ≥ 5 $(= \mathcal{B})$

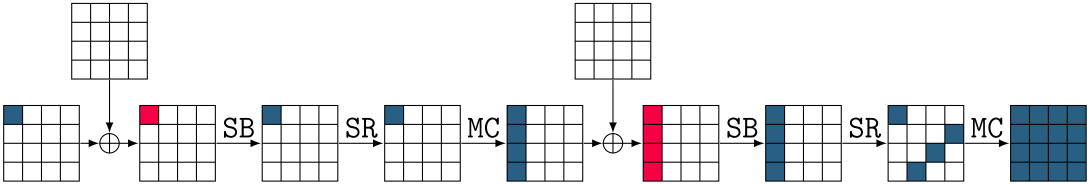
MILP – Example application: AES [MWGP11; WW11]



Variables: 1 binary variable per state byte (active/inactive)

- AddRoundKey: input $=$ output
- SubBytes: input $=$ output, cost $=$ sum(inputs)
- ShiftRows: variable renaming
- MixColumns: for each active column: sum(inputs) $+$ sum(outputs) ≥ 5 $(= \mathcal{B})$

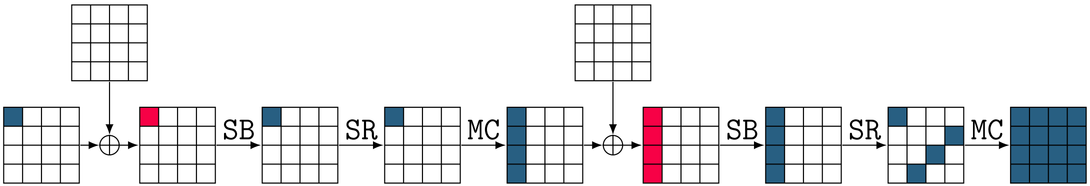
MILP – Example application: AES [MWGP11; WW11]



Variables: 1 binary variable per state byte (active/inactive)

- AddRoundKey: input = output
- SubBytes: input = output, cost = sum(inputs)
- ShiftRows: variable renaming
- MixColumns: for each active column: sum(inputs) + sum(outputs) $\geq$ 5 (= $\mathcal{B}$)

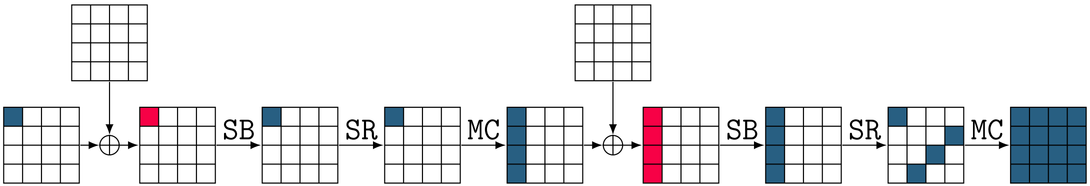
MILP – Example application: AES [MWGP11; WW11]



Variables: 1 binary variable per state byte (active/inactive)

- AddRoundKey: input = output
- SubBytes: input = output, cost = sum(inputs)
- ShiftRows: variable renaming
- MixColumns: for each active column: sum(inputs) + sum(outputs) $\geq$ 5 ($= \mathcal{B}$)

MILP – Example application: AES [MWGP11; WW11]



Variables: 1 binary variable per state byte (active/inactive)

- AddRoundKey: input = output
- SubBytes: input = output, cost = sum(inputs)
- ShiftRows: variable renaming
- MixColumns: for each active column: sum(inputs) + sum(outputs) ≥ 5 $(= \mathcal{B})$

MILP – Example application: AES [MWGP11]

Variables:

- $S_{r,i,j} \in \{0, 1\}$: Is S-box in row i, column j in round r active?
- $M_{r,j} \in \{0, 1\}$: Is MixColumns j in round r active?

Linear Program:

$$\min \sum_{r,i,j} S_{r,i,j} \qquad \text{(Min \# active S-boxes)}$$

$$\text{s.t.} \quad \mathcal{B} \cdot M_{r,j} \leq \sum_i S_{r,i,(i+j)\%4} + \sum_i S_{r+1,i,j} \leq 8 \cdot M_{r,j} \qquad \text{(For each MixColumns)}$$

$$\sum_{i,j} S_{0,i,j} \geq 1 \qquad \text{(Non-triviality)}$$

MILP – Example application: AES [MWGP11]

Variables:

- $S_{r,i,j} \in \{0,1\}$: Is S-box in row i, column j in round r active?
- $M_{r,j} \in \{0,1\}$: Is MixColumns j in round r active?

Linear Program:

$$\min \sum_{r,i,j} S_{r,i,j} \qquad \text{(Min \# active S-boxes)}$$

$$\text{s.t.} \quad \mathcal{B} \cdot M_{r,j} \leq \sum_{i} S_{r,i,(i+j)\%4} + \sum_{i} S_{r+1,i,j} \leq 8 \cdot M_{r,j} \qquad \text{(For each MixColumns)}$$

$$\sum_{i,j} S_{0,i,j} \geq 1 \qquad \text{(Non-triviality)}$$

MILP – Example application: AES – Code in sagemath

```
#!/usr/bin/env sage
rounds = range(4)
p = MixedIntegerLinearProgram(maximization=False)
S = p.new_variable(name='sbox', binary=True)
M = p.new_variable(name='mcol', binary=True)

for r in rounds:
    for j in [0..3]:
        activecells = sum(S[r,i,(i+j)%4] for i in [0..3]) \
                    + sum(S[r+1,i,j] for i in [0..3])
        p.add_constraint(5*M[r,j] <= activecells <= 8*M[r,j])
p.add_constraint(sum(S[0,i,j] for i in [0..3] for j in [0..3]) >= 1)

p.set_objective(sum(S[r,i,j] for r in rounds for i in [0..3] \
                                              for j in [0..3]))
p.solve()
print(p.get_objective_value(), p.get_values(S))
```

MILP – Advanced models

- Modeling more complex relations of “allowed transitions” accurately
  - Activity patterns for linear layers: XOR, near-MDS matrices [ABD+23], …
  - DDT/LAT for bitwise S-box models [SHW+14b; SHW+14a], ARX [FWG+16]
- Need to translate **vertex representation** into **half-space representation**

XOR :

- inputs ☐, ☐ → output ☐
- inputs ☐, ■ → output ■
- inputs ■, ☐ → output ■
- inputs ■, ■ → output ☐ or ■

➔

$$\begin{cases} I_1 + I_2 \geq O \\ I_1 + O \geq I_2 \\ I_2 + O \geq I_1 \end{cases}$$

- For large tables, this becomes very heavy (e.g., 8-bit S-boxes [AST+17; SW23])

SAT/SMT/CP – Different Levels of Convenience

1. **SAT (Satisfiability) Solvers**: Find valid solution or prove unsatisfiability of CNF

$$\bigwedge_i \bigvee_j \ell_{i,j} \qquad \text{with literals } \ell_{i,j} \in \{v_{i,j}, \neg v_{i,j}\}$$

   Example solvers: `MiniSAT`, `lingeling`, and a myriad others

2. **SMT (Sat. Modulo Theories) Solvers**: Accept a more general grammar including bitvector operations such as integer addition.

   Example solvers: `STP` ("Simple Theorem Prover"), …

3. **CP (Constraint Programming) Solvers**: Accept an even more general grammar (depends on solver).

   Example solvers/frameworks: `MiniZinc`, `Z3`, `Choco`, …

SAT/SMT/CP – Different Levels of Convenience

1. **SAT (Satisfiability) Solvers**: Find valid solution or prove unsatisfiability of CNF

$$\bigwedge_i \bigvee_j \ell_{i,j} \qquad \text{with literals } \ell_{i,j} \in \{v_{i,j}, \neg v_{i,j}\}$$

   Example solvers: `MiniSAT`, `lingeling`, and a myriad others

2. **SMT (Sat. Modulo Theories) Solvers**: Accept a more general grammar including bitvector operations such as integer addition.

   Example solvers: `STP` ("Simple Theorem Prover"), …

3. **CP (Constraint Programming) Solvers**: Accept an even more general grammar (depends on solver).

   Example solvers/frameworks: `MiniZinc`, `Z3`, `Choco`, …

SAT/SMT/CP – Different Levels of Convenience

1. **SAT (Satisfiability) Solvers**: Find valid solution or prove unsatisfiability of CNF

$$\bigwedge_i \bigvee_j \ell_{i,j} \qquad \text{with literals } \ell_{i,j} \in \{v_{i,j}, \neg v_{i,j}\}$$

   Example solvers: `MiniSAT`, `lingeling`, and a myriad others

2. **SMT (Sat. Modulo Theories) Solvers**: Accept a more general grammar including bitvector operations such as integer addition.

   Example solvers: `STP` ("Simple Theorem Prover"), …

3. **CP (Constraint Programming) Solvers**: Accept an even more general grammar (depends on solver).

   Example solvers/frameworks: `MiniZinc`, `Z3`, `Choco`, …

SAT/SMT/CP for Finding Distinguishers [MP13; Köl14]

- Solves a **constraint satisfaction problem**, may not be optimal
  - “Emulate” optimization: “is there a solution better than $X, X+1, X+2, \ldots$?”
- Useful to find valid solutions under some constraints
  - Finding characteristics that follow a given truncated pattern
  - Finding solutions for other crypto problems (preimage, …)
- Not so efficient for some more complex problems
  - Not so good for modelling a **cost** sum or optimization [EME22]
  - Not perfectly **parallelizable**

Dedicated Algorithms

Dedicated Tools for Hash Functions: Examples

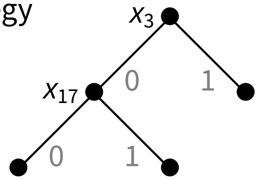
- **SHA-1**: `HashClash`
  https://github.com/cr-marcstevens/hashclash

- **SHA-2**: `nldtool`
  https://github.com/iaikkrypto/nldtool

- **SHA-3**: `KeccakTools`
  https://github.com/KeccakTeam/KeccakTools

Dedicated Guess-and-Determine Search

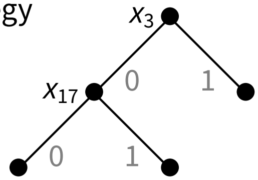
- Guess-and-Determine Search is a general search strategy
  - Traverse search tree to find a valid solution
  - SAT solvers use it on CNF level
  - This is an example on small (differential) circuits



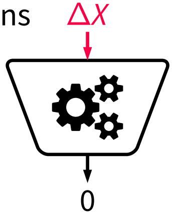
- `nldtool`: Automated search for characteristics and solutions
  - Hash collision search
  - Application example: SHA-2 [MNS11; MNS13; DEM15]

Dedicated Guess-and-Determine Search

- Guess-and-Determine Search is a general search strategy
  - Traverse search tree to find a valid solution
  - SAT solvers use it on CNF level
  - This is an example on small (differential) circuits



- `nldtool`: Automated search for characteristics and solutions
  - Hash collision search
  - Application example: `SHA-2` [MNS11; MNS13; DEM15]



Guess-and-Determine Search Algorithm

while there are undetermined bits **do**

Decision (Guessing)

1. Pick an undetermined bit
2. Constrain this bit

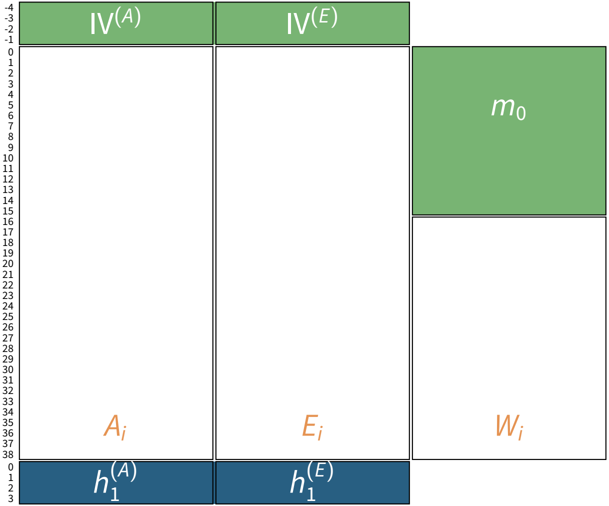
Deduction (Propagating)

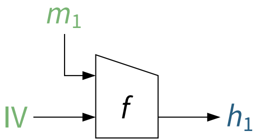
3. Propagate the new information to other variables and equations
4. **if** no inconsistency is detected, goto step 1

Correction (Backtracking)

5. **if** possible, apply a different constraint to this bit, goto step 3
6. **else** undo guesses until this critical bit can be resolved

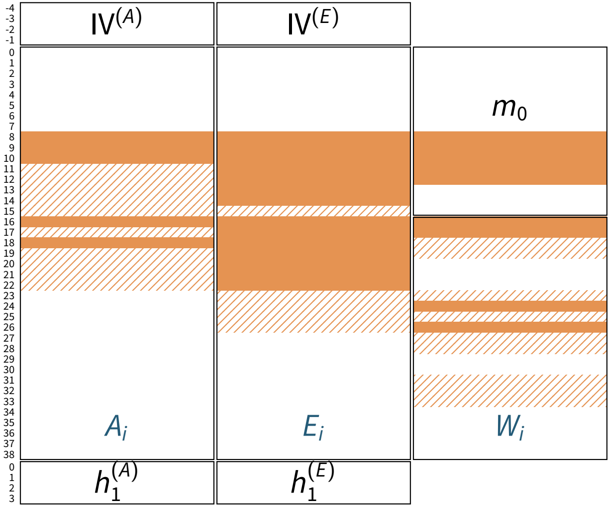
Example: Semi-Free-Start Collision for 39-step SHA-512





- Shows state words A_i, E_i, W_i
- Inputs IV, m_1
- Output h_1

Example: Semi-Free-Start Collision for 39-step SHA-512

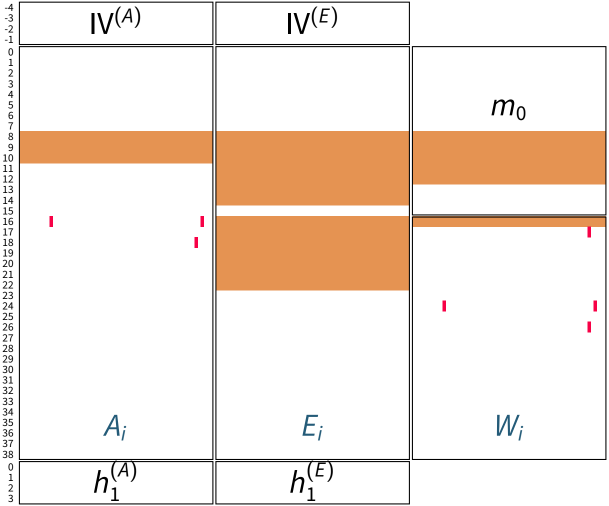


Starting point:

0. "Local Collision" with few active message words

- Active words with differences [?]
- No differences [-] (cancellation required)
- No differences [-]

Example: Semi-Free-Start Collision for 39-step SHA-512

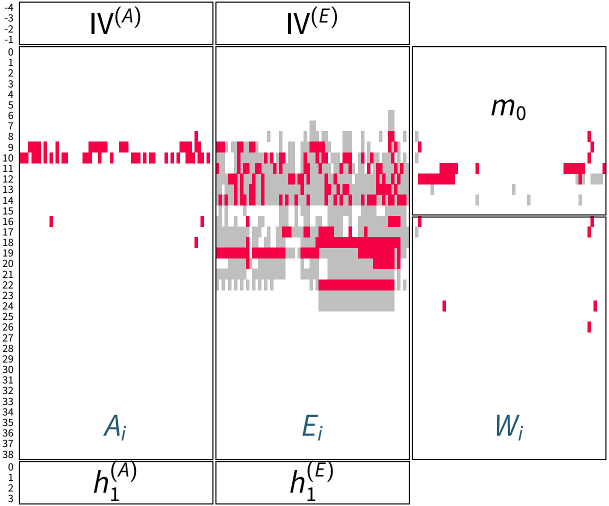


Search strategy:

1. Fix high-probability parts
2. Fix signed differences
3. Find message pair

- Active words with some differences [?]
- Active bits [n,u,x]
- Inactive bits [-]
- Fixed inactive bits [0,1]

Example: Semi-Free-Start Collision for 39-step SHA-512

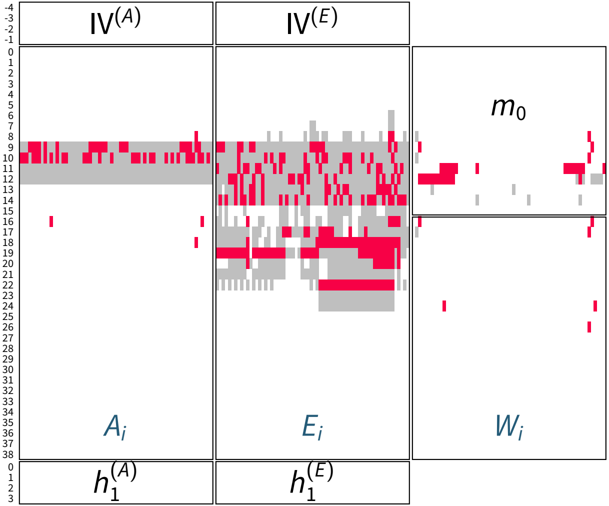


Search strategy:

1. Fix high-probability parts
2. Fix signed differences
3. Find message pair

- Active words with some differences [?]
- Active bits [n,u]
- Inactive bits [-]
- Fixed inactive bits [0,1]

Example: Semi-Free-Start Collision for 39-step `SHA-512`

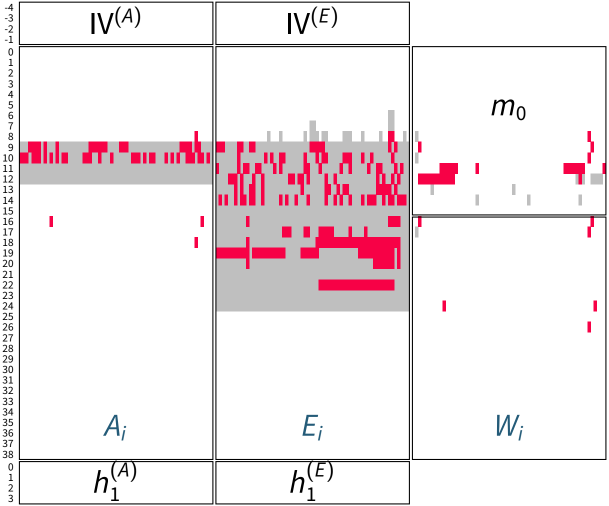


Search strategy:

1. Fix high-probability parts
2. Fix signed differences
3. Find message pair

- Active words with some differences [?]
- Active bits [n,u]
- Inactive bits [-]
- Fixed inactive bits [0,1]

Example: Semi-Free-Start Collision for 39-step SHA-512

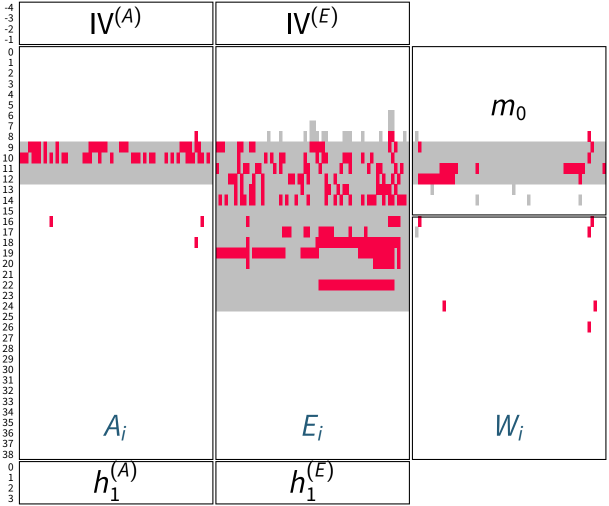


Search strategy:

1. Fix high-probability parts
2. Fix signed differences
3. Find message pair

- Active words with some differences [?]
- Active bits [n,u]
- Inactive bits [-]
- Fixed inactive bits [0,1]

Example: Semi-Free-Start Collision for 39-step SHA-512

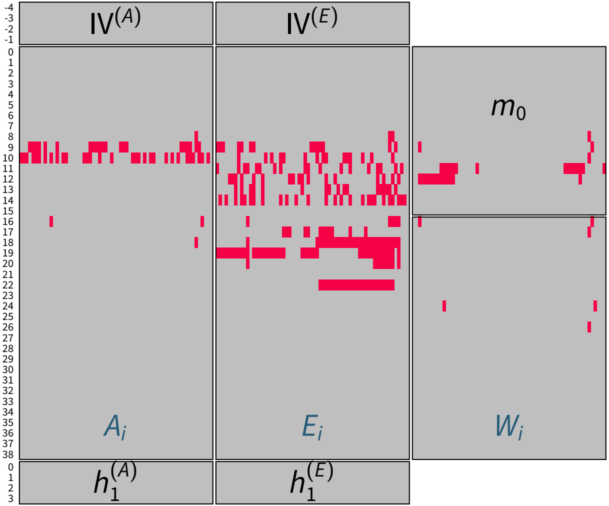


Search strategy:

1. Fix high-probability parts
2. Fix signed differences
3. Find message pair

- Active words with some differences [?]
- Active bits [n,u]
- Inactive bits [-]
- Fixed inactive bits [0,1]

Example: Semi-Free-Start Collision for 39-step SHA-512



Search strategy:

1. Fix high-probability parts
2. Fix signed differences
3. Find message pair

- Active words with some differences [?]
- Active bits [n,u]
- Inactive bits [-]
- Fixed inactive bits [0,1]

Optimized Key Recovery Attacks

The Need for Tools

- Key recovery has long been ignored
- Fewer choices to make for the attacker

…but…

- Optimizations involve choices and tradeoffs
- Precise evaluation is tedious
- “Optimal” distinguisher doesn’t guarantee optimal attack

The Need for Tools

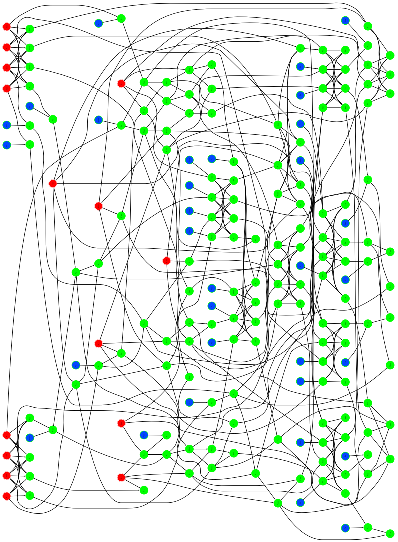
- Key recovery has long been ignored
- Fewer choices to make for the attacker

…but…

- Optimizations involve choices and tradeoffs
- Precise evaluation is tedious
- “Optimal” distinguisher doesn’t guarantee optimal attack

Optimizing (Only) the Key-Recovery Steps: Examples

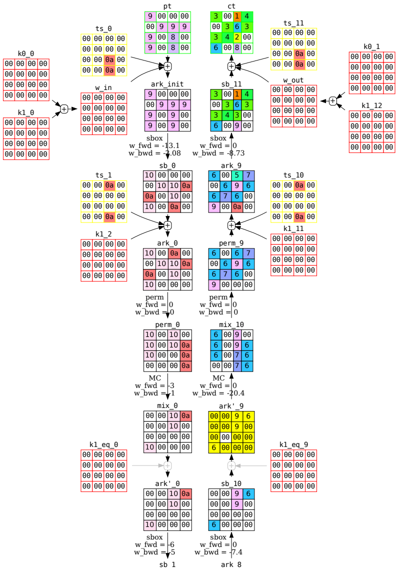
- **Guess-and-determine** attacks:
  - Autoguess [HE22a] [a]



[a] https://github.com/hadipourh/autoguess

Optimizing (Only) the Key-Recovery Steps: Examples

- **Guess-and-determine** attacks:
  - Autoguess [HE22a] [a]
- **Differential** cryptanalysis:
  - keyrecoverytool [Nag22] [b]



[a] https://github.com/hadipourh/autoguess

[b] https://extgit.iaik.tugraz.at/castle/tool/keyrecoverytool

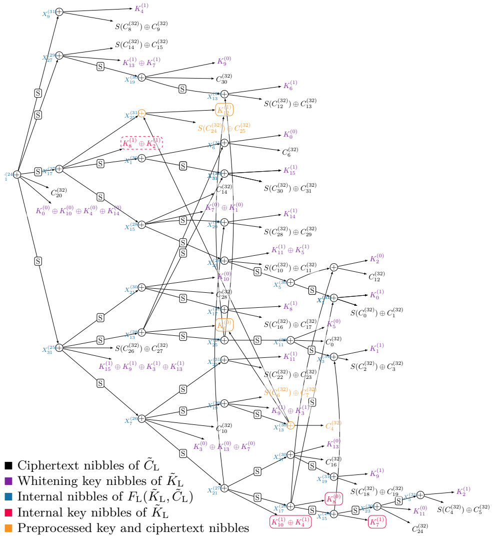
Optimizing (Only) the Key-Recovery Steps: Examples

- **Guess-and-determine** attacks:
  - **Autoguess** [HE22a] [a]
- **Differential** cryptanalysis:
  - **keyrecoverytool** [Nag22] [b]
  - **KYRYDI** [BDD+24] [c]
- Integral cryptanalysis:

[a] https://github.com/hadipourh/autoguess
[b] https://extgit.iaik.tugraz.at/castle/tool/keyrecoverytool
[c] https://gitlab.inria.fr/capsule/kyrydi

Optimizing (Only) the Key-Recovery Steps: Examples

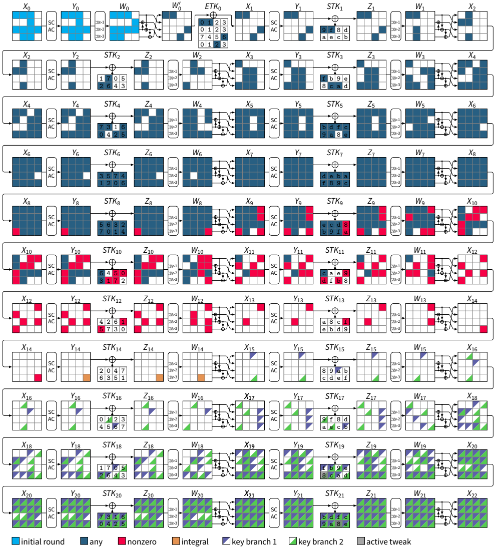
- **Guess-and-determine** attacks:
  - Autoguess [HE22a] [a]
- **Differential** cryptanalysis:
  - keyrecoverytool [Nag22] [b]
  - KYRYDI [BDD+24] [c]
- **Integral** cryptanalysis:
  - Graph-based [HE22b] [d]
  - AutoPSy [HSE23] [e]



[a] https://github.com/hadipourh/autoguess
[b] https://extgit.iaik.tugraz.at/castle/tool/keyrecoverytool
[c] https://gitlab.inria.fr/capsule/kyrydi
[d] https://github.com/hadipourh/mpt
[e] https://github.com/hadipourh/zero

Optimizing (Only) the Key-Recovery Steps: Examples

- **Guess-and-determine** attacks:
  - Autoguess [HE22a] [a]
- **Differential** cryptanalysis:
  - keyrecoverytool [Nag22] [b]
  - KYRYDI [BDD+24] [c]
- **Integral** cryptanalysis:
  - Graph-based [HE22b] [d]
  - AutoPSy [HSE23] [e]



[a] https://github.com/hadipourh/autoguess
[b] https://extgit.iaik.tugraz.at/castle/tool/keyrecoverytool
[c] https://gitlab.inria.fr/capsule/kyrydi
[d] https://github.com/hadipourh/mpt
[e] https://github.com/hadipourh/zero

Why Optimizing Full Key Recovery Attacks is Challenging

- Preferably use a **joint model** for distinguisher and key recovery
  - Only works for satisfiability-based distinguishers
- **Complexity formulas** are often complicated
  - Mix of polynomial/exponential terms; simplified assumptions
- **Multi-step** processes lead to heavy models
- Very different types of **key schedules**
- Many different **optimizations** and strategies

Why Optimizing Full Key Recovery Attacks is Challenging

- Preferably use a **joint model** for distinguisher and key recovery
  - Only works for satisfiability-based distinguishers
- **Complexity formulas** are often complicated
  - Mix of polynomial/exponential terms; simplified assumptions
- **Multi-step** processes lead to heavy models
- Very different types of **key schedules**
- Many different **optimizations** and strategies

Why Optimizing Full Key Recovery Attacks is Challenging

- Preferably use a **joint model** for distinguisher and key recovery
  - Only works for satisfiability-based distinguishers
- **Complexity formulas** are often complicated
  - Mix of polynomial/exponential terms; simplified assumptions
- **Multi-step** processes lead to heavy models
- Very different types of **key schedules**
- Many different **optimizations** and strategies

Why Optimizing Full Key Recovery Attacks is Challenging

- Preferably use a **joint model** for distinguisher and key recovery
  - Only works for satisfiability-based distinguishers
- **Complexity formulas** are often complicated
  - Mix of polynomial/exponential terms; simplified assumptions
- **Multi-step** processes lead to heavy models
- Very different types of **key schedules**
- Many different **optimizations** and strategies

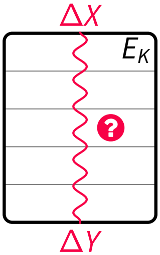
Why Optimizing Full Key Recovery Attacks is Challenging

- Preferably use a **joint model** for distinguisher and key recovery
  - Only works for satisfiability-based distinguishers
- **Complexity formulas** are often complicated
  - Mix of polynomial/exponential terms; simplified assumptions
- **Multi-step** processes lead to heavy models
- Very different types of **key schedules**
- Many different **optimizations** and strategies

Impossibility-based Distinguishers

Some distinguishers are based on the **non-existence** of a valid characteristic:

- **Differential** > Impossible differentials
- **Linear** > Zero-correlation linear approximations
- **Integral** > Division/monomial trail; ZC-based integrals

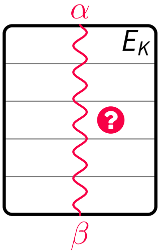


However, models for full attacks need **solution-based** distinguisher models
(or a quantified language like QSAT)

Impossibility-based Distinguishers

Some distinguishers are based on the **non-existence** of a valid characteristic:

- **Differential** > Impossible differentials
- **Linear** > Zero-correlation linear approximations
- **Integral** > Division/monomial trail; ZC-based integrals

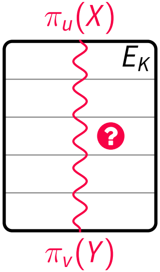


However, models for full attacks need **solution-based** distinguisher models
(or a quantified language like QSAT)

Impossibility-based Distinguishers

Some distinguishers are based on the **non-existence** of a valid characteristic:

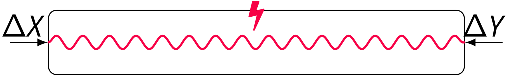
- **Differential** › Impossible differentials
- **Linear** › Zero-correlation linear approximations
- **Integral** › Division/monomial trail; ZC-based integrals



However, models for full attacks need **solution-based** distinguisher models
(or a quantified language like QSAT)

Two Ways of Modelling Impossibility

Unsatisfiability-based:



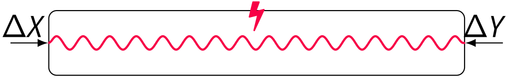
- First specify distinguisher, then check
- Precise, but potentially slow

Satisfiability-based:

- Find distinguisher that misses in the middle
- Typically efficient, but less precise

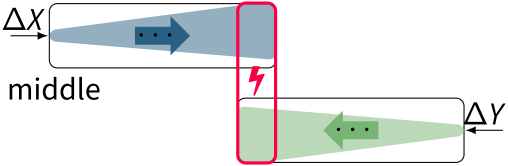
Two Ways of Modelling Impossibility

Unsatisfiability-based:



- First specify distinguisher, then check
- Precise, but potentially slow

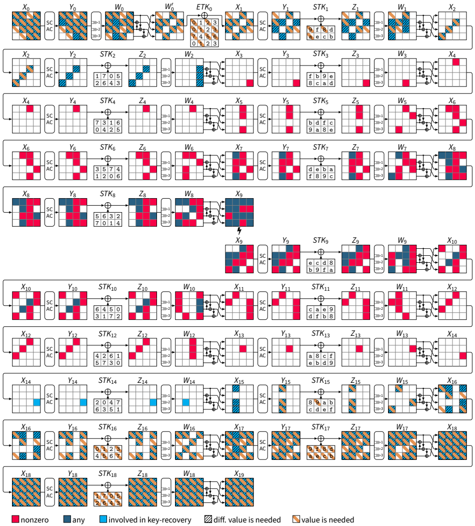
Satisfiability-based:



- Find distinguisher that misses in the middle
- Typically efficient, but less precise

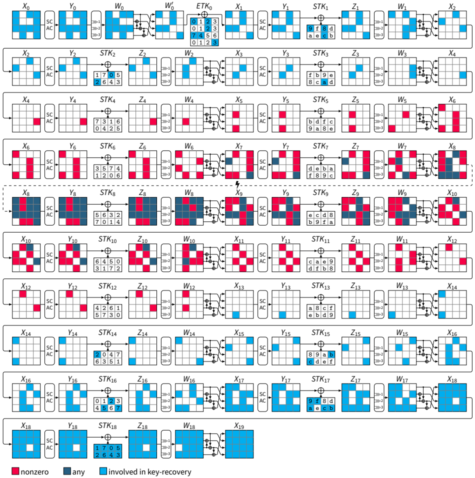
Example: Finding Full ID/ZC/Integral Attacks [HSE23]

- **Impossible-differential (ID) attacks**
- Zero-correlation (ZC) attacks
- ZC-based integral attacks*



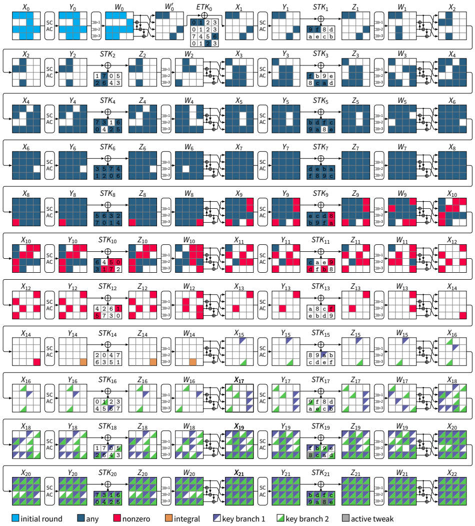
Example: Finding Full ID/ZC/Integral Attacks [HSE23]

- **Impossible-differential (ID) attacks**
- **Zero-correlation (ZC) attacks**
- ZC-based integral attacks*



Example: Finding Full ID/ZC/Integral Attacks [HSE23]

- Impossible-differential (ID) attacks
- Zero-correlation (ZC) attacks
- ZC-based integral attacks*



Frameworks

What exactly is a “Framework”?

Judging from paper titles, we have a plethora of frameworks, but …

- ? Generality & Applicability
- ? Reuseability & Extensibility
- ? Maintainability & Verifiability

Frameworks: Examples

- **CryptoSMT** [Köl14; AK18] https://github.com/kste/cryptosmt
  - Differential/linear trails, clustering, key/preimage recovery, ...; based on SMT (STP, Boolector, CryptoMiniSat)
- **CASCADA** [RR22] https://github.com/ranea/CASCADA
  - Differential/linear trails, impossible differentials/zero-correlation, ...; based on SMT
- **CLAASP** [BGG+23] https://github.com/Crypto-TII/claasp
  - All of the above, neural tests; supports many solvers

Frameworks: Challenges

- **Cipher representation**
  - Based on building blocks? As a DAG? Software/hardware code?
- **Efficiency vs. precision**
- **Simplicity vs. dedicated optimizations**
- **Meta-challenges**: Conflicting incentives in academia

Frameworks: Challenges

- **Cipher representation**
  - Based on building blocks? As a DAG? Software/hardware code?
- **Efficiency vs. precision**
- **Simplicity vs. dedicated optimizations**
- **Meta-challenges**: Conflicting incentives in academia

Conclusion

- Tools and solvers can help find attacks and derive bounds

- Very active area with many open challenges
  - More efficient and precise models
  - Application to other design paradigms and attack techniques
  - Modeling full attacks (not just the distinguisher)
  - Frameworks and reuseability

Bibliography I

[ABD+23] Roberto Avanzi, Subhadeep Banik, Orr Dunkelman, Maria Eichlseder, Shibam Ghosh, Marcel Nageler, and Francesco Regazzoni. **The QARMAv2 Family of Tweakable Block Ciphers.** IACR Trans. Symmetric Cryptol. 2023.3 (2023), pp. 25–73. DOI: `10.46586/TOSC.V2023.I3.25-73`.

[AK18] Ralph Ankele and Stefan Kölbl. **Mind the Gap - A Closer Look at the Security of Block Ciphers against Differential Cryptanalysis.** SAC 2018. Vol. 11349. LNCS. Springer, 2018, pp. 163–190. DOI: `10.1007/978-3-030-10970-7_8`.

[AST+17] Ahmed Abdelkhalek, Yu Sasaki, Yosuke Todo, Mohamed Tolba, and Amr M. Youssef. **MILP Modeling for (Large) S-boxes to Optimize Probability of Differential Characteristics.** IACR Transactions on Symmetric Cryptology 2017.4 (2017), pp. 99–129. DOI: `10.13154/tosc.v2017.i4.99-129`.

[BDD+24] Christina Boura, Nicolas David, Patrick Derbez, Rachelle Heim Boissier, and María Naya-Plasencia. **A generic algorithm for efficient key recovery in differential attacks - and its associated tool.** IACR Cryptol. ePrint Arch. (2024), p. 288. URL: `https://eprint.iacr.org/2024/288`.

[BGG+23] Emanuele Bellini, David Gérault, Juan Grados, Yun Ju Huang, Rusydi H. Makarim, Mohamed Rachidi, and Sharwan K. Tiwari. **CLAASP: A Cryptographic Library for the Automated Analysis of Symmetric Primitives.** SAC 2023. Vol. 14201. LNCS. Springer, 2023, pp. 387–408. DOI: `10.1007/978-3-031-53368-6_19`.

Bibliography II

[BS90] Eli Biham and Adi Shamir. **Differential Cryptanalysis of DES-like Cryptosystems.** Advances in Cryptology – CRYPTO 1990. Vol. 537. LNCS. Springer, 1990, pp. 2–21. DOI: 10.1007/3-540-38424-3_1.

[DEM15] Christoph Dobraunig, Maria Eichlseder, and Florian Mendel. **Analysis of SHA-512/224 and SHA-512/256.** Advances in Cryptology – ASIACRYPT 2015. Vol. 9453. LNCS. Springer, 2015, pp. 612–630. DOI: 10.1007/978-3-662-48800-3_25.

[EME22] Johannes Erlacher, Florian Mendel, and Maria Eichlseder. **Bounds for the Security of Ascon against Differential and Linear Cryptanalysis.** IACR Trans. Symmetric Cryptol. 2022.1 (2022), pp. 64–87. DOI: 10.46586/TOSC.V2022.I1.64-87. URL: https://doi.org/10.46586/tosc.v2022.i1.64-87.

[FWG+16] Kai Fu, Meiqin Wang, Yinghua Guo, Siwei Sun, and Lei Hu. **MILP-Based Automatic Search Algorithms for Differential and Linear Trails for Speck.** Fast Software Encryption – FSE 2016. Vol. 9783. LNCS. Springer, 2016, pp. 268–288. DOI: 10.1007/978-3-662-52993-5_14.

[HE22a] Hosein Hadipour and Maria Eichlseder. **Autoguess: A Tool for Finding Guess-and-Determine Attacks and Key Bridges.** ACNS 2022. Vol. 13269. LNCS. Springer, 2022, pp. 230–250. DOI: 10.1007/978-3-031-09234-3_12.

[HE22b] Hosein Hadipour and Maria Eichlseder. **Integral Cryptanalysis of WARP based on Monomial Prediction.** IACR Trans. Symmetric Cryptol. 2022.2 (2022), pp. 92–112. DOI: 10.46586/TOSC.V2022.I2.92-112.

Bibliography III

[HSE23] Hosein Hadipour, Sadegh Sadeghi, and Maria Eichlseder. **Finding the Impossible: Automated Search for Full Impossible-Differential, Zero-Correlation, and Integral Attacks.** EUROCRYPT 2023. Vol. 14007. LNCS. Springer, 2023, pp. 128–157. DOI: 10.1007/978-3-031-30634-1_5.

[Knu94] Lars R. Knudsen. **Truncated and Higher Order Differentials.** Fast Software Encryption – FSE 1994. Vol. 1008. LNCS. Springer, 1994, pp. 196–211. DOI: 10.1007/3-540-60590-8_16.

[Köl14] Stefan Kölbl. **CryptoSMT: An easy to use tool for cryptanalysis of symmetric primitives.** 2014. URL: https://github.com/kste/cryptosmt.

[KW02] Lars R. Knudsen and David Wagner. **Integral Cryptanalysis.** Fast Software Encryption – FSE 2002. Vol. 2365. LNCS. Springer, 2002, pp. 112–127. DOI: 10.1007/3-540-45661-9_9.

[Lai94] Xuejia Lai. **Higher Order Derivatives and Differential Cryptanalysis.** Communications and Cryptography: Two Sides of One Tapestry. Vol. 276. International Series in Engineering and Computer Science. Kluwer Academic Publishers, 1994, pp. 227–233. DOI: 10.1007/978-1-4615-2694-0_23.

[Mat93] Mitsuru Matsui. **Linear Cryptanalysis Method for DES Cipher.** Advances in Cryptology – EUROCRYPT 1993. Vol. 765. LNCS. Springer, 1993, pp. 386–397. DOI: 10.1007/3-540-48285-7_33.

Bibliography IV

[MNS11] Florian Mendel, Tomislav Nad, and Martin Schläffer. **Finding SHA-2 Characteristics: Searching through a Minefield of Contradictions.** Advances in Cryptology – ASIACRYPT 2011. Vol. 7073. LNCS. Springer, 2011, pp. 288–307. DOI: 10.1007/978-3-642-25385-0_16.

[MNS13] Florian Mendel, Tomislav Nad, and Martin Schläffer. **Improving Local Collisions: New Attacks on Reduced SHA-256.** Advances in Cryptology – EUROCRYPT 2013. Vol. 7881. LNCS. Springer, 2013, pp. 262–278. DOI: 10.1007/978-3-642-38348-9_16.

[MP13] Nicky Mouha and Bart Preneel. **Towards Finding Optimal Differential Characteristics for ARX: Application to Salsa20.** IACR Cryptology ePrint Archive, Report 2013/328. 2013.

[MWGP11] Nicky Mouha, Qingju Wang, Dawu Gu, and Bart Preneel. **Differential and Linear Cryptanalysis Using Mixed-Integer Linear Programming.** Information Security and Cryptology – Inscrypt 2011. Vol. 7537. LNCS. Springer, 2011, pp. 57–76. DOI: 10.1007/978-3-642-34704-7_5.

[Nag22] Marcel Nageler. **Automatic cryptanlysis of block ciphers: Finding efficient key-recovery attacks.** MA thesis. Graz University of Technology, 2022. DOI: 10.3217/n8ehm-dgj71.

[RR22] Adrián Ranea and Vincent Rijmen. **Characteristic automated search of cryptographic algorithms for distinguishing attacks (CASCADA).** IET Inf. Secur. 16.6 (2022), pp. 470–481. DOI: 10.1049/ISE2.12077. URL: https://doi.org/10.1049/ise2.12077.

Bibliography V

[SHW+14a] Siwei Sun, Lei Hu, Meiqin Wang, Peng Wang, Kexin Qiao, Xiaoshuang Ma, Danping Shi, Ling Song, and Kai Fu. **Towards Finding the Best Characteristics of Some Bit-oriented Block Ciphers and Automatic Enumeration of (Related-key) Differential and Linear Characteristics with Predefined Properties.** IACR Cryptology ePrint Archive, Report 2014/747. 2014.

[SHW+14b] Siwei Sun, Lei Hu, Peng Wang, Kexin Qiao, Xiaoshuang Ma, and Ling Song. **Automatic Security Evaluation and (Related-key) Differential Characteristic Search: Application to SIMON, PRESENT, LBlock, DES(L) and Other Bit-Oriented Block Ciphers.** Advances in Cryptology – ASIACRYPT 2014. Vol. 8873. LNCS. Springer, 2014, pp. 158–178. DOI: 10.1007/978-3-662-45611-8_9.

[SW23] Ling Sun and Meiqin Wang. **SoK: Modeling for Large S-boxes Oriented to Differential Probabilities and Linear Correlations.** IACR Trans. Symmetric Cryptol. 2023.1 (2023), pp. 111–151. DOI: 10.46586/TOSC.V2023.I1.111-151.

[WW11] Shengbao Wu and Mingsheng Wang. **Security Evaluation against Differential Cryptanalysis for Block Cipher Structures.** IACR Cryptology ePrint Archive, Report 2011/551. 2011.