Tools for Cryptanalysis

Maria Eichlseder

Applied Cryptography 2 – ST 2020

SCIENCE
PASSION
TECHNOLOGY

Outline

Computer-Aided Cryptanalysis

Tools for Finding Differential Characteristics

- Method 1: Matsui's branch-and-bound algorithm
- Method 2: Mixed-Integer Linear Programming (MILP)
- Method 3: Boolean Satisfiability and Constraint Programming (SAT/SMT, CP)
- Method 4: Dedicated Guess-and-Determine search

Computer-Aided Cryptanalysis

Pen-and-Paper or Computer?

- Many published attacks are presented and (mostly) verifiable via pen-and-paper
- Many published attacks are round-reduced attacks on “good ciphers” with impractical complexity (at least for university budgets) since their main purpose is to evaluate the security margin conservatively: Better overestimate than underestimate the attacker!

- Experimental evaluation of parts of the attack is important
- Ciphers are designed to be complex: some problems infeasible to solve by hand
- Designing ciphers is also complex: use tools to find good building blocks

Computer-Aided Cryptanalysis – Examples

- Linear & Differential Cryptanalysis
  - Finding good characteristics for many rounds
  - Proving bounds for the best possible characteristics
  - Finding right pairs, particularly for hash collisions
- Advanced Linear & Differential Attacks
  - Finding “combinable” fragments (impossible diff., diff.-lin., …)
- Algebraic Attacks
  - Equation solving for key recovery
  - Finding algebraic properties over many rounds (cubes, division property…)

Tools for Finding Differential Characteristics

Automated Tools for (Differential) Cryptanalysis

Motivation:

Finding good characteristics can be hard, but is necessary to evaluate new designs

Solvers:

By hand

General-purpose solvers:

- SAT/SMT (Boolean SATisfiability/Sat. Modulo Theories)
- MILP (Mixed Integer Linear Programming)
- CP (Constraint Programming):

Dedicated solvers

- Matsui's branch-and-bound algorithm
- `KeccakTools` (SHA-3), `nltool` (SHA-2), ...
- ...

Basic Approach

Model constraints that characterize correct characteristics/solutions

- Coarse-grained: truncated patterns (which S-boxes are active?)
- Fine-grained: precise differences/masks

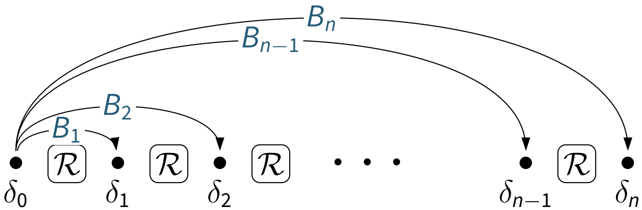
Model cost (if applicable)

Express the search goal: any one / all / best / good solution(s)?

Method 1: Matsui's Branch-and-Bound algorithm

- Introduced by Matsui to find the best characteristics for DES [Mat94]
- A dynamic programming technique working round-by-round:
  - B_i – the best (highest) differential probability for i rounds
  - $\overline{B_i}$ – a lower bound $\overline{B_i} \leq B_i$, e.g., the probability of *some* characteristic

Idea: Derive the best n-round probability B_n from knowing the best i-round probabilities B_i $(1 \leq i \leq n-1)$



Method 1: Matsui's Branch-and-Bound algorithm

The algorithm works by induction over the number of rounds n:

1. To initialize $\overline{B_n}$, iteratively extend the best $(n-1)$-round characteristic by 1 round $\rightarrow \overline{B_n} = B_{n-1} \cdot p_n$
2. For B_n, traverse the search tree and cut bad branches:
   - `round1`: For each $\delta_0 \rightarrow \delta_1$, if $p_1 B_{n-1} \geq \overline{B_n}$: call `round2`
   - `round2`: For each $\delta_1 \rightarrow \delta_2$, if $p_1 p_2 B_{n-2} \geq \overline{B_n}$: call `round3`
   - ...
   - If $p = p_1 p_2 \cdots p_n \geq \overline{B_n}$, update $\overline{B_n} := p$

Method 1: Matsui's Branch-and-Bound algorithm – Properties

- Solves an optimization problem, finds best characteristic

- Efficient when there are not too many candidates to test:
  - Small state size, few good characteristics
  - Lightweight Feistel ciphers may be good candidates
  - Partial results can be combined to get bounds (without solutions)

- Not feasible for many modern ciphers
  - State size too large, too many good characteristics:
    Need to iterate over all $\delta_0 \to \delta_1$ etc.
  - Not trivial to adapt for related-key characteristics etc.

Method 2: Mixed-Integer Linear Program (MILP)

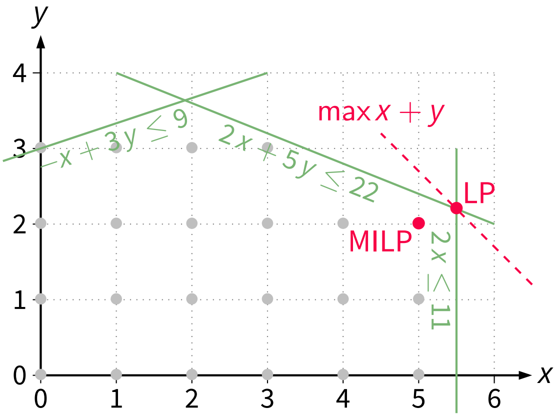
Linear Programming (LP) is a method to solve optimization problems

- 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 $1 \leq j \leq J$:

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

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

Method 2: LP vs. MILP



Method 2: MILP – Solvers

Hardness of LP/MILP solving:

- LP: Efficient solving algorithms such as Dantzig's simplex method are available.
- MILP problems can be NP-hard. Solvers combine LP solvers with branch-and-bound.

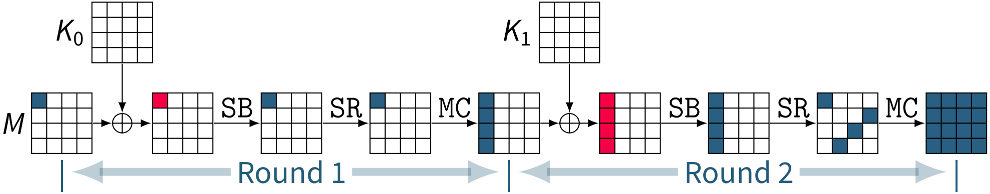
Some well-known solver software:

- `IBM ILOG CPLEX`
- `Gurobi`

These solvers can be used as stand-alone software (`.lp` input files)
or as libraries with convenient interfaces (`C/C++`, `sagemath`, ...).

Method 2: MILP – Example application: AES

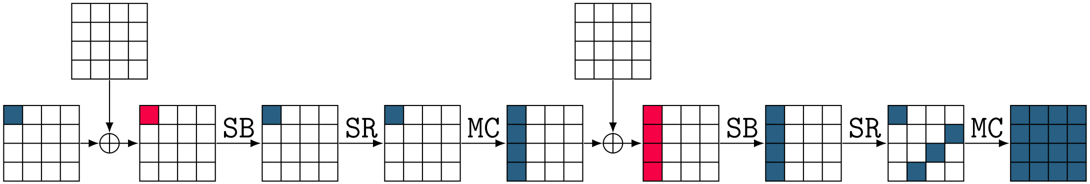
Idea: First find (one of) the best “truncated patterns” with MILP:



Then find exact differences with some other means – or just derive a bound from the number of active S-boxes:

$$\underbrace{\begin{bmatrix}00&00&00&40\\00&00&00&00\\00&00&00&00\\00&00&00&00\end{bmatrix} \xmapsto{\texttt{SB}} \begin{bmatrix}00&00&00&6a\\00&00&00&00\\00&00&00&00\\00&00&00&00\end{bmatrix}}_{2^{-6}} \xmapsto{\texttt{SR}} \begin{bmatrix}00&00&00&6a\\00&00&00&00\\00&00&00&00\\00&00&00&00\end{bmatrix} \xmapsto{\substack{\texttt{MC}\\\texttt{AK}}} \underbrace{\begin{bmatrix}00&00&00&d4\\00&00&00&6a\\00&00&00&6a\\00&00&00&be\end{bmatrix} \xmapsto{\texttt{SB}} \begin{bmatrix}00&00&00&2b\\00&00&00&61\\00&00&00&61\\00&00&00&cd\end{bmatrix}}_{2^{-6\times 4}} \xmapsto{\texttt{SR}} \begin{bmatrix}00&00&00&2b\\00&00&61&00\\00&61&00&00\\cd&00&00&00\end{bmatrix} \xmapsto{\substack{\texttt{SR}\\\texttt{MC}}} \begin{bmatrix}cd&61&a3&56\\cd&a3&c2&2b\\4c&c2&61&2b\\81&61&61&7d\end{bmatrix}$$

Method 2: MILP – Example application: AES [MWGP11]



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 $(= \mathcal{B})$

Method 2: 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:

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

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

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

Method 2: 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))
```

Method 2: MILP – Example application: AES

- For any $k \cdot 4$ rounds, results confirm $k \cdot 25$ active S-boxes from theory
- Model is more interesting for related-key characteristics where there may be differences in the key
  - `AddRoundKey`: Model key-`xor` by its branch number $\mathcal{B} = 2$
  - Model key schedule (similar operations)
  - Result: Fewer active S-boxes!
- Similar approaches quite popular for new designs, particularly tweakable block ciphers ($\rightarrow$ related-tweakey!)

Method 2: MILP – Advanced models 1

- Bitwise Boolean functions: Some ciphers combine AES-like operations with some bitwise operations, such as XOR (key/tweak schedule, Feistel, ...).

  There are many useful gadgets in MILP modelling for translating Boolean expressions (like $A \vee B$ or $A \Rightarrow B$) to MILP conditions, but we are interested in the bytewise differential/linear patterns of these operations, e.g.:

$$\texttt{XOR}: \begin{cases} \text{inputs } \square, \square & \rightarrow \text{ output } \square \\ \text{inputs } \square, \blacksquare & \rightarrow \text{ output } \blacksquare \\ \text{inputs } \blacksquare, \square & \rightarrow \text{ output } \blacksquare \\ \text{inputs } \blacksquare, \blacksquare & \rightarrow \text{ output } \square \text{ or } \blacksquare \end{cases} \qquad \texttt{AND}: \begin{cases} \text{inputs } \square, \square & \rightarrow \text{ output } \square \\ \text{inputs } \square, \blacksquare & \rightarrow \text{ output } \square \text{ or } \blacksquare \\ \text{inputs } \blacksquare, \square & \rightarrow \text{ output } \square \text{ or } \blacksquare \\ \text{inputs } \blacksquare, \blacksquare & \rightarrow \text{ output } \square \text{ or } \blacksquare \end{cases}$$

  These can be modeled by their branch number (XOR: model $\mathcal{B} = 2$) or by writing Boolean conditions & translating (AND: $O \Rightarrow I_1 \vee I_2$, in MILP: $O \leq I_1 + I_2$)

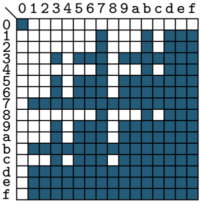
Method 2: MILP – Advanced models 2

- Lightweight MixColumns: Many lightweight ciphers use more lightweight `MixColumns` matrices with smaller branch number $\mathcal{B}$, for example

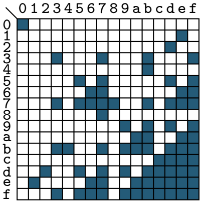
$$\mathtt{MC} = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \qquad \text{with} \quad \mathcal{B} = 4 \qquad (\text{“near-MDS”})$$

Simple models using $\mathcal{B} = 4$ or a sequence of `XOR`s allow too many patterns:

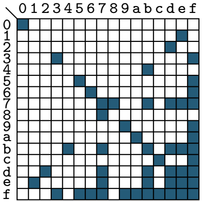
(truncated columns in hex notation; e.g., $5 = (0, 1, 0, 1)^\top$) [DEKM16]



(a) Branch number model



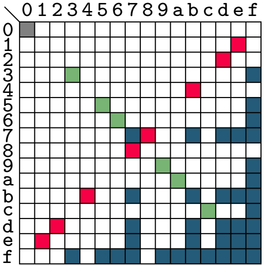
(b) `XOR` model



(c) Exact model

Method 2: MILP – Advanced models 2

The valid patterns $(a_0, \ldots, a_3)^\top \to (b_0, \ldots, b_3)^\top$ are exactly the following:



0 active cells

4 active cells and $a_i = b_i$

4 active cells and $a_i = b_i \oplus 1$

$\Bigg\}\ \forall i : a_i \oplus b_i = \bigoplus_j a_j$

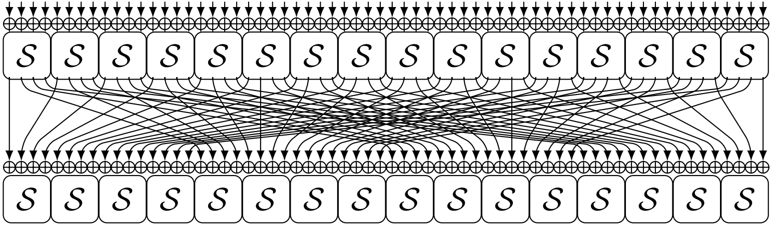
6 or more active cells $\to \sum_i a_i + \sum_i b_i \geq 6$

We need to express $(C_0 \wedge C_1 \wedge C_2 \wedge C_3) \vee C_4$ ($C_{0\ldots3}$ for bit i of 0 4 4, C_4 for 6):

- 5 binary helper variables $C_0 \ldots C_4$ and constraint $C_0 + C_1 + C_2 + C_3 + 4\,C_4 \geq 4$
- Condition C_4: constraint $\sum_i a_i + \sum_i b_i \geq 6\,C_4$
- Conditions $C_{0\ldots3}$: integer helper X for `XOR`, constraint $b_i + \sum_{j \neq i} a_j = 2\,X + (1 - C_i)$

Method 2: MILP – Advanced models 3

- S-box details and bitwise models: Sometimes, patterns of active S-boxes are not sufficient for good bounds – think of `ARX` ciphers or `PRESENT`:



This would require a linear model of the S-box DDT, which is usually more complex than the previous `MixColumns` example, particularly for large S-boxes.

There are tools/algorithms that can perform such a translation
Vertex representation (table) $\rightarrow$ Half-space representation (linear inequalities)

Method 2: MILP – Properties

- Solves an optimization problem

- Useful to prove bounds for “strongly aligned”, `AES`-like ciphers
  - Cost evaluation as a (weighted) sum works nicely
  - Can have more complex cost metrics using weights
  - Often works very efficiently even for full-round ciphers

- Not so useful for complex bitwise descriptions and characteristics
  - Language of linear inequalities is not so natural for crypto
  - Too many integer variables lead to bad solver performance

Method 3: SAT/SMT/CP – Different Levels of Convenience

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

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

   Any set of Boolean constraints can (and needs to) be translated to CNF.

   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. Solvers often translate these into CNF and feed the result to SAT solvers.

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

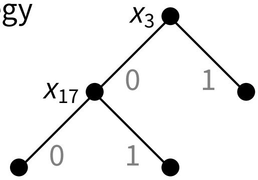
3. CP (Constraint Programming) Solvers: Accept an even more general grammar (depends on solver). Example solvers: `Z3`, `Choco`, …

Method 3: SAT/SMT/CP – Properties

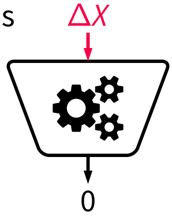
- Solves a satisfiability problem, may not be optimal
  - "Emulate" optimization: "is there a solution better than $X, X+1, X+2, \ldots$?"
- Useful to find valid solutions under some constraints
  - Finding characteristics that follow a given truncated pattern
  - Finding solutions for other crypto problems (preimage, ...)
  - Well-suited for modelling the Boolean networks in ciphers (except `XOR`s)
- Not so efficient for some more complex problems
  - Not too many rounds (too many variables!)
  - Not so good for modelling a cost sum or optimization

Method 4: 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



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



Generalized Differences: Motivation [DR06]

1. ARX designs with modular addition and XOR of w-bit words:

   bitwise signed difference $\Delta^{\pm} \in \{0, +1, -1\}^w$

   uniquely determines both

   modular difference $\Delta^{\boxminus} = \sum_i \Delta_i^{\pm} 2^i \in \mathbb{Z}_{2^w}$

   bitwise xor difference $\Delta^{\oplus} = (|\Delta_i^{\pm}|)_i$

2. Search progress: Represent all stages of the evolution from a starting point (where only some zero-differences are fixed) via the characteristic (of signed differences) to the message pair (of fixed bit values).

Generalized Differences

Let (x_j, x_j^*) be a pair of bits. The generalized condition $\nabla(x_j, x_j^*)$ constrains the possible values of (x_j, x_j^*) to a subset of all pairs $\{(1,1),(0,1),(1,0),(0,0)\}$

$\bullet$ = is-allowed and $\circ$ = is-not-allowed

$\nabla(z_j, z_j^*)$	$\nabla(z_j, z_j^*)$	$\nabla(z_j, z_j^*)$	$\nabla(z_j, z_j^*)$
`0` = ○○○●	`-` = ●○○●	`3` = ○○●●	`7` = ○●●●
`u` = ○○●○	`x` = ○●●○	`5` = ○●○●	`B` = ●○●●
`n` = ○●○○	`#` = ○○○○	`A` = ●○●○	`D` = ●●○●
`1` = ●○○○	`?` = ●●●●	`C` = ●●○○	`E` = ●●●○

Search: start from undetermined bits and refine until all are $\in \{\texttt{0}, \texttt{1}, \texttt{n}, \texttt{u}\}$

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

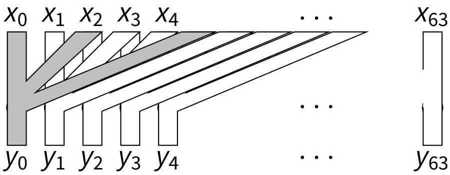
Correction (Backtracking)

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

Bitsliced Propagation 1

Divide the crypto circuit into small “bitslices” with few involved bits

Example: Linear layer



Example: Modular addition $z = x + y \mod 2^w \Rightarrow z_i = x_i \oplus y_i \oplus c_i$ with carry bit c_i

Bitsliced Propagation 2

Example: Bit-`XOR` operation $y = f(x) = x_1 \oplus x_2$ with

$$\nabla(x, x^*) = [\texttt{?x}], \qquad \nabla(y, y^*) = [\texttt{-}], \qquad \text{or in short,} \quad \nabla(z, z^*) = [\texttt{?x-}].$$

This generalized difference allows 16 (of $4^3 = 64$) solutions $(z, z^*) = (x_1 x_2 y, x_1^* x_2^* y^*)$:

$$\begin{array}{cccc} (000, 010), & (000, 110), & (100, 010), & (100, 110), \\ (010, 000), & (010, 100), & (110, 000), & (110, 100), \\ (001, 011), & (001, 111), & (101, 011), & (101, 111), \\ (011, 001), & (011, 101), & (111, 001), & (111, 101). \end{array}$$

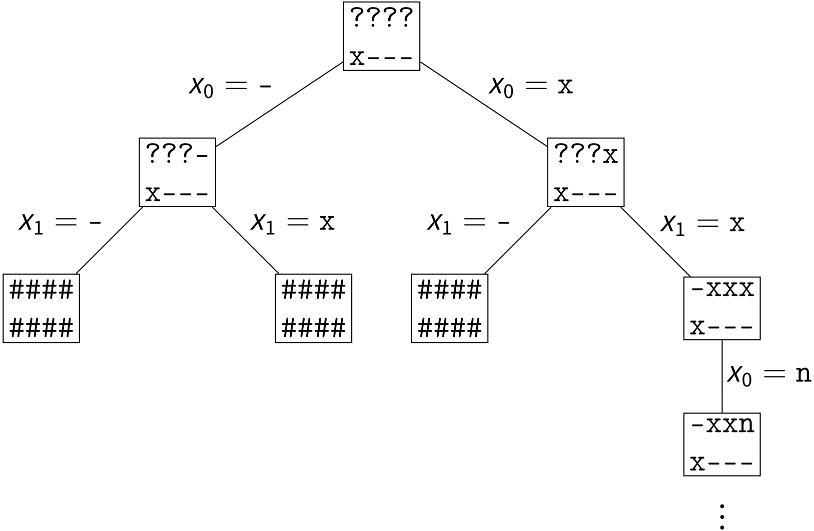
However, only 4 of the 12 are valid input and output combinations for f:

$$(000, 110), \quad (110, 000), \quad (101, 011), \quad (011, 101).$$

The minimal generalized difference that contains all the above true solutions is

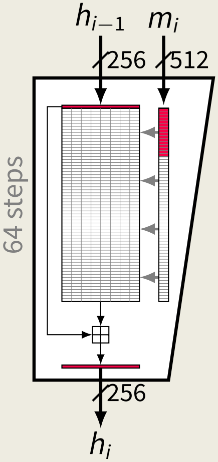
$$\nabla(z, z^*) = [\texttt{xx-}].$$

Branching & Backtracking

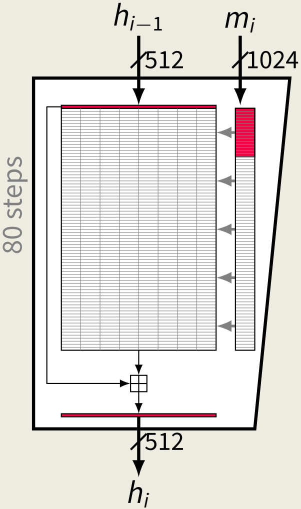


Example: SHA-2 – Compression Function

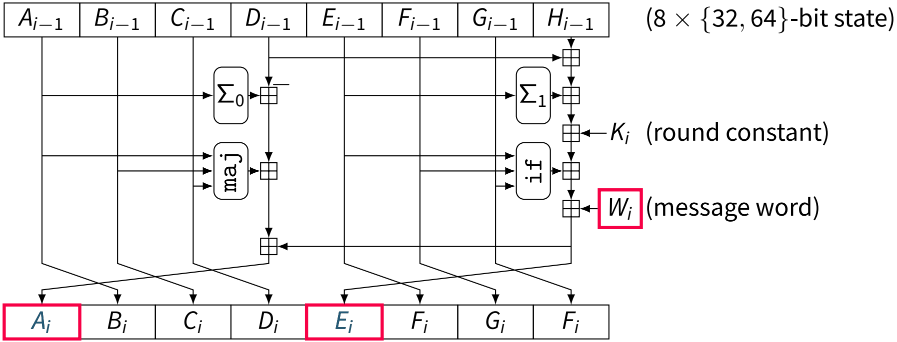
SHA-256



SHA-512



Example: SHA-2 – Round Function (64 or 80 Rounds)



$$\mathtt{maj}(x,y,z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)$$

$$\mathtt{if}(x,y,z) = (x \wedge y) \oplus (\neg x \wedge z)$$

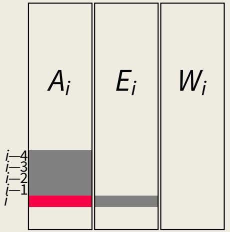
$$\Sigma_0(x) = (x \ggg 2) \oplus (x \ggg 13) \oplus (x \ggg 22) \qquad \text{(for SHA-256)}$$

$$\Sigma_1(x) = (x \ggg 6) \oplus (x \ggg 11) \oplus (x \ggg 25) \qquad \text{(for SHA-256)}$$

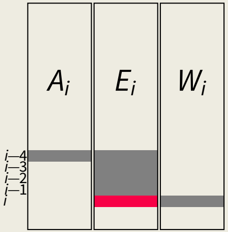
Example: `SHA-2` – Round Function, Alternative Representation

Recursive update patterns for ■ A_i, E_i, and W_i using ■:

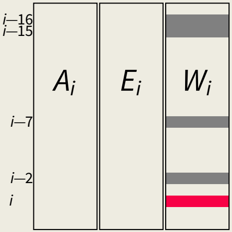
A_i update



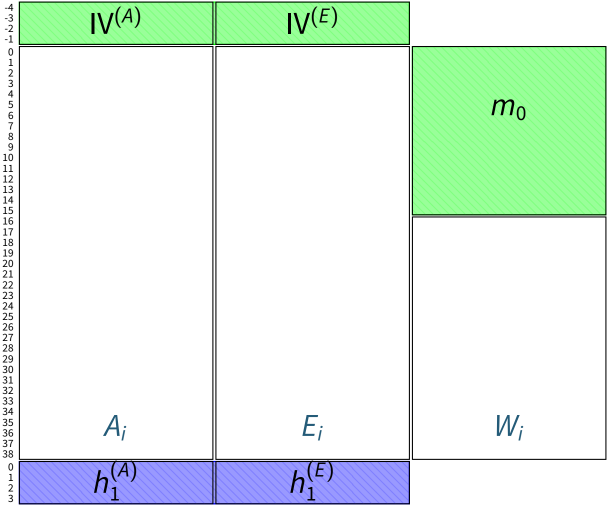
E_i update

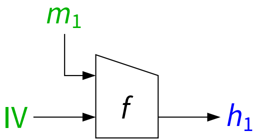


W_i expansion



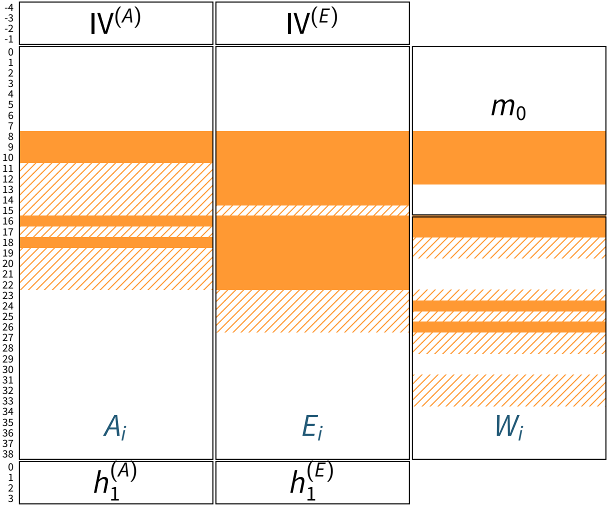
Example: Semi-Free-Start Collision for 39 / 80 steps of `SHA-512`





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

Example: Semi-Free-Start Collision for 39 / 80 steps of SHA-512

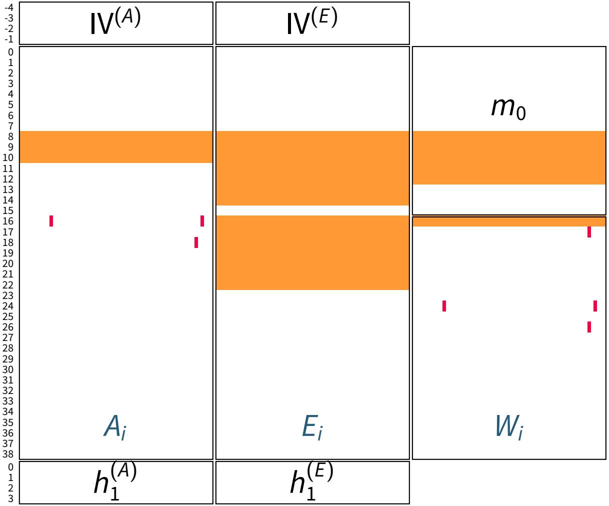


Starting point:

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 / 80 steps of SHA-512

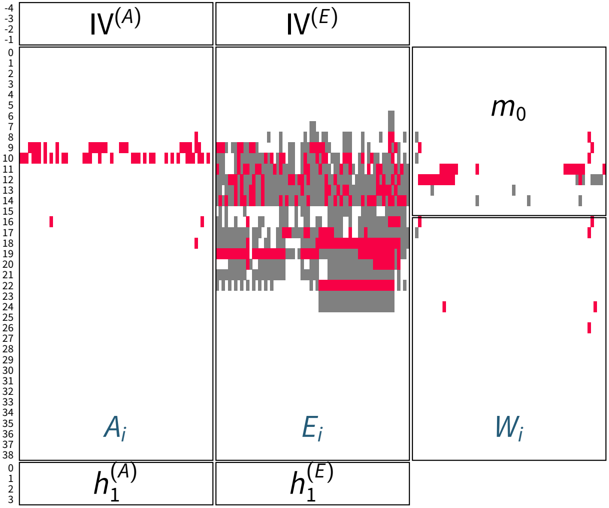


Search strategy:

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 / 80 steps of SHA-512

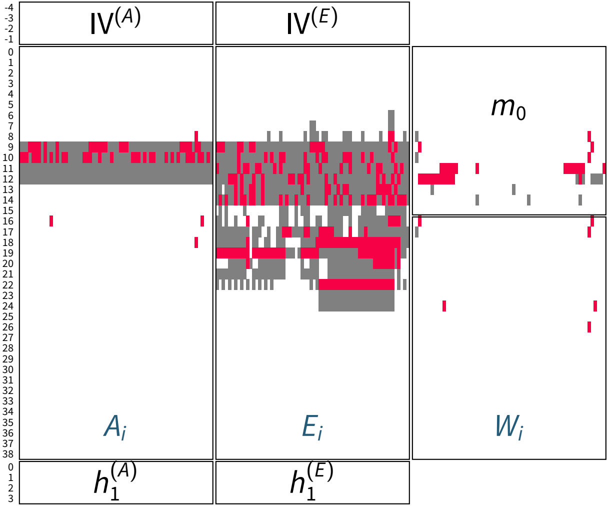


Search strategy:

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 / 80 steps of `SHA-512`

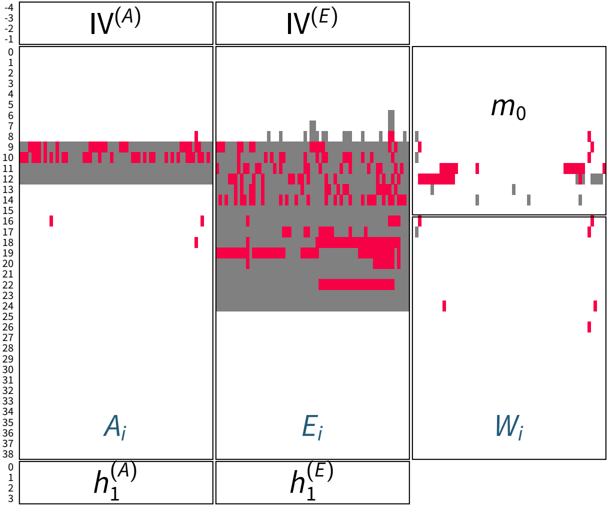


Search strategy:

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 / 80 steps of `SHA-512`

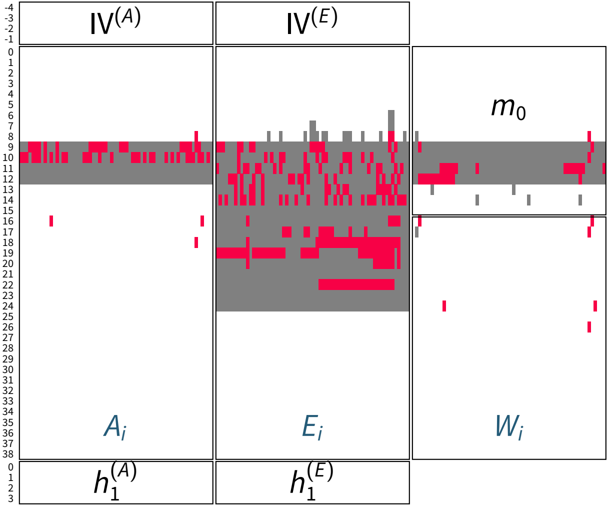


Search strategy:

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 / 80 steps of SHA-512

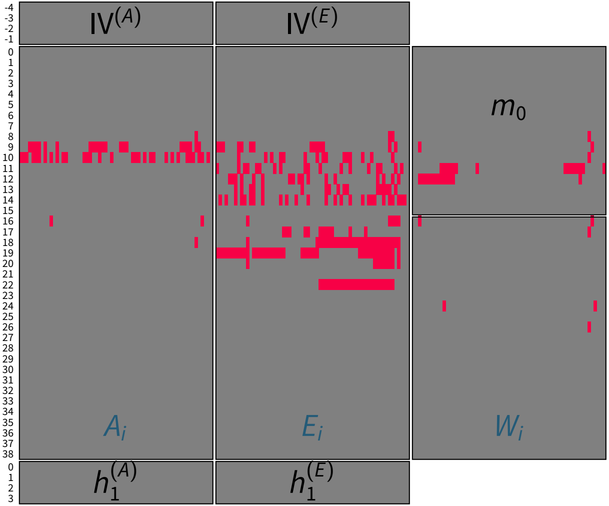


Search strategy:

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 / 80 steps of 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]

Improving Guess & Determine?

- Problem description
  - Starting point and high-level strategy
  - Hash function description
- Guessing strategy, branching rules
  - Which variable to pick first? Which value to guess first for this variable?
- Propagation
  - How to determine implications of a guess?
  - How to detect contradictions?
- Backtracking
  - How many guesses to undo? When to restart?

Conclusion

Computer-Aided Cryptanalysis

Tools for Finding Differential Characteristics

- Method 1: Matsui's branch-and-bound algorithm
- Method 2: Mixed-Integer Linear Programming (MILP)
- Method 3: Boolean Satisfiability and Constraint Programming (SAT/SMT, CP)
- Method 4: Dedicated Guess-and-Determine search

Questions

Questions you should be able to answer

1. What are the respective advantages/disadvantages of searching characteristics by hand, using general-purpose solvers, or using dedicated solvers?
2. Explain Matui's Branch-and-Bound algorithm and discuss its advantages/disadvantages.
3. Model the problem of bounding the number of active S-boxes of `AES` as a Mixed-Integer Linear Program (MILP). Explain the model.
4. Outline the dedicated guess-and-determine search algorithm discussed in the lecture, and explain how it propagates information.

Bibliography I

[DEKM16] Christoph Dobraunig, Maria Eichlseder, Daniel Kales, and Florian Mendel. **Practical Key-Recovery Attack on MANTIS5.** IACR Transactions on Symmetric Cryptology 2016.2 (2016), pp. 248–260. DOI: 10.13154/tosc.v2016.i2.248-260.

[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.

[DR06] Christophe De Cannière and Christian Rechberger. **Finding SHA-1 Characteristics: General Results and Applications.** Advances in Cryptology – ASIACRYPT 2006. Vol. 4284. LNCS. Springer, 2006, pp. 1–20. DOI: 10.1007/11935230_1.

[Mat94] Mitsuru Matsui. **On Correlation Between the Order of S-boxes and the Strength of DES.** Advances in Cryptology – EUROCRYPT ’94. Vol. 950. LNCS. Springer, 1994, pp. 366–375. DOI: 10.1007/BFb0053451.

Bibliography II

[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.

[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.