Artificial Intelligence methods for the design of cryptographic primitives

Luca Mariot
L.Mariot@tudelft.nl

AICrypt@EUROCRYPT 2021
Zagreb, October 16, 2021
Intro – AI in symmetric crypto

AI-based optimization methods in cryptography

AI-based computational models in cryptography

Conclusions
This talk is based on the chapter:

Intro – AI in symmetric crypto

AI-based optimization methods in cryptography

AI-based computational models in cryptography

Conclusions
Symmetric ciphers require several low-level primitives, such as:

- Pseudorandom number generators (PRNG)
- Boolean functions $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2$ and S-boxes
- Permutation (diffusion) layers, ...
AI approaches to design symmetric primitives

- "Traditional" approach: ad-hoc and algebraic constructions to choose primitives with specific security properties
- "AI" approach: support the designer in choosing the primitives using AI methods/models from the following domains:
  - Optimization (Evolutionary algorithms, swarm intelligence...)
  - Computational models (cellular automata, neural networks...)

\[ F : \{0, 1\}^n \rightarrow \{0, 1\}^m \]

Luca Mariot
Artificial Intelligence methods for the design of cryptographic primitives
Intro – AI in symmetric crypto

AI-based optimization methods in cryptography

AI-based computational models in cryptography

Conclusions
Combinatorial Optimization

- **Combinatorial Optimization Problem**: map $\mathcal{P} : \mathcal{I} \rightarrow S$ from a set $\mathcal{I}$ of problem instances to a family $S$ of solution spaces.

- $S = \mathcal{P}(\mathcal{I})$ is a finite set equipped with a *fitness function* $\text{fit} : S \rightarrow \mathbb{R}$, giving a score to candidate solutions $x \in S$.

- **Optimization goal**: find $x^* \in S$ such that:

  - **Minimization**: $x^* = \text{argmin}_{x \in S} \{ \text{fit}(x) \}$
  - **Maximization**: $x^* = \text{argmax}_{x \in S} \{ \text{fit}(x) \}$

- **Heuristic optimization algorithm**: iteratively *tweaks* a set of candidate solutions using $\text{fit}$ to drive the search.
Evolutionary Algorithms (EA) – Genetic Algorithms (GA)

Optimization algorithms loosely based on evolutionary principles, introduced respectively by J. Holland (1975) and J. Koza (1989)

- Work on a **coding** of the candidate solutions
- Evolve in parallel a **population** of solutions.
- **Black-box optimization**: use only the fitness function to optimize the solutions.
- Use **Probabilistic operators** to evolve the solutions

**GA Encoding**: Typically, an individual is represented with a fixed-length bitstring

\[
0 1 1 1 1 0 0 0
\]

\[
f(x_1, x_2, x_3) = x_1 \cdot x_2 \oplus x_1 \oplus x_2 \oplus x_3
\]
GP Encoding: an individual is represented by a tree

- Terminal nodes: input variables of a program
- Internal nodes: operators (e.g. AND, OR, NOT, XOR, ...)

\[ f(x_1, x_2, x_3, x_4) = (x_1 \ AND \ x_2) \ OR \ (x_3 \ XOR \ x_4) \]
The EA Loop

Initialize Population → Selection → Crossover → Mutation → Fitness Evaluation → Replace → Terminate? (Yes) → Output Best Solution (No) → Initialize Population → ...
Several design steps can be cast as combinatorial optimization problems, such as the search of:

- **Boolean functions** \( f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2 \) for stream ciphers

  ![Diagram of LFSR and Boolean function]

  - LFSR 1 \( \rightarrow x_1 \)
  - LFSR 2 \( \rightarrow x_2 \)
  - ... \( \rightarrow ... \)
  - LFSR \( n \rightarrow x_n \)

  \[ f(x_1, x_2, \ldots, x_n) \rightarrow \text{next bit} \]

- **S-Boxes** \( F : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^m \) for block ciphers

Possible advantages of using EA for this search:

- **Diversity** of solutions, due to the "blindness" of EA
- **Flexibility** of EA (optimizing several properties at once)
Several properties to consider for thwarting attacks, e.g.:

A **Boolean function** used in the combiner model should:
- be balanced
- have high algebraic degree $d$
- have high nonlinearity $nl(F)$
- be resilient of high order $t$

A $(n, n)$-**function** used in the SPN paradigm should
- be balanced ($\leftrightarrow$ bijective)
- have high nonlinearity $N_F$
- have low differential uniformity $\delta_F$
Constructions of good Boolean Functions and S-Boxes

- Number of Boolean functions of $n$ variables: $2^{2^n}$

<table>
<thead>
<tr>
<th>$n$</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2^{2^n}$</td>
<td>256</td>
<td>65536</td>
<td>4.3 · $10^9$</td>
<td>1.8 · $10^{19}$</td>
<td>3.4 · $10^{38}$</td>
<td>1.2 · $10^{77}$</td>
</tr>
</tbody>
</table>

- ⇒ too huge for exhaustive search when $n > 5$!

In practice, one usually resorts to:

- **Algebraic constructions** (*Maiorana-McFarland, Rothaus,...*) [Carlet21]
- **Combinatorial optimization techniques**
  - *Simulated Annealing* [Clark04]
  - *Evolutionary Algorithms* [Millan98, Picek16]
  - *Swarm Intelligence* [M15] [Mariot15], ...
Evolving Boolean Functions with GA

- GA encoding: represent the truth tables as $2^n$-bit strings
- Fitness function: combines nonlinearity, algebraic degree, correlation-immunity
- Specialized crossover and mutation operators for preserving balancedness

**Crossover Idea:** Use *counters* to keep track of the multiplicities of zeros and ones [Millan98, Manzoni20]

```
\[ p_1 \quad 0 \ 1 \ 0 \ 1 \ 0 \ 1 \ 1 \ 0 \]
\[ p_2 \quad 1 \ 0 \ 0 \ 0 \ 1 \ 0 \ 1 \ 1 \]
```

\[ \chi \Rightarrow \begin{array}{ccccccccc} 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ \text{count}[1] = 4 & \text{fill with 0} & c \end{array} \]
Evolving Boolean Functions with GP

- The truth table is synthesized from a GP tree:

```
\[ f(x) = x_1 \land (x_2 \lor \neg x_3) \]
```

```
\begin{array}{ccc|c}
  x_1 & x_2 & x_3 & f(x) \\
  0 & 0 & 0 & 0 \\
  0 & 0 & 1 & 0 \\
  0 & 1 & 0 & 1 \\
  0 & 1 & 1 & 0 \\
  1 & 0 & 0 & 1 \\
  1 & 0 & 1 & 0 \\
  1 & 1 & 0 & 0 \\
  1 & 1 & 1 & 0 \\
\end{array}
```

- Difficult to enforce constraints on balancedness with crossover and mutation.
GP and its variants generally fares better than GA on optimizing Boolean functions [P16]


Similar results to traditional algebraic constructions
Intro – AI in symmetric crypto

AI-based optimization methods in cryptography

AI-based computational models in cryptography

Conclusions
One-dimensional Cellular Automaton (CA): a discrete parallel computation model composed of a finite array of $n$ cells.

Example: $n = 6$, $d = 3$, $\omega = 0$, $f(s_i, s_{i+1}, s_{i+2}) = s_i \oplus s_{i+1} \oplus s_{i+2}$ (rule 150)

No Boundary CA – NBCA

Periodic Boundary CA – PBCA

Each cell updates its state $s \in \{0, 1\}$ by applying a local rule $f : \{0, 1\}^d \rightarrow \{0, 1\}$ to itself, the $\omega$ cells on its left and the $d - 1 - \omega$ cells on its right.
Motivations

**General Research Goal:** Investigate cryptographic primitives defined by Cellular Automata

Why CA, anyway?

1. **Security from Complexity:** CA can yield very complex dynamical behaviors, depending on the local rule

2. **Efficient implementation:** Leverage CA parallelism and locality for lightweight cryptography
CA-based Pseudorandom Generator (PRG) [Wolfram86]: central cell of rule 30 CA used as a stream cipher keystream

Security claims based mainly on statistical/empirical tests

This CA-based PRNG was later shown to be vulnerable, improvements by choosing larger local rules [Leporati14]
Local rule: $\chi(x_1, x_2, x_3) = x_1 \oplus (1 \oplus (x_2 \cdot x_3))$ (rule 210)

Invertible for every odd size $n$ of the CA

Used as a PBCA with $n = 5$ in the Keccak specification of SHA-3 standard [Keccak11]
Goal: Find PBCA of length \( n \) and diameter \( d = n \):
- with cryptographic properties on par with those of other real-world ciphers [Mariot19]
- with low implementation cost [Picek17]

Considered S-boxes sizes: from \( n = 4 \) to \( n = 8 \)

Genetic Programming to address this problem

**Fitness function:** optimize both crypto (nonlinearity, differential uniformity) and implementation properties (GE measure)
### Results

**Table:** Statistical results and comparison.

<table>
<thead>
<tr>
<th>S-box size</th>
<th>$T_{max}$</th>
<th>GP</th>
<th>$N_F$</th>
<th>$\delta_F$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 x 4</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>4</td>
</tr>
<tr>
<td>5 x 5</td>
<td>42</td>
<td>42</td>
<td>41.73</td>
<td>1.01</td>
</tr>
<tr>
<td>6 x 6</td>
<td>86</td>
<td>84</td>
<td>80.47</td>
<td>4.72</td>
</tr>
<tr>
<td>7 x 7</td>
<td>182</td>
<td>182</td>
<td>155.07</td>
<td>8.86</td>
</tr>
<tr>
<td>8 x 8</td>
<td>364</td>
<td>318</td>
<td>281.87</td>
<td>13.86</td>
</tr>
</tbody>
</table>

- From $n = 4$ to $n = 7$, one obtains CA rules inducing S-boxes with optimal crypto properties.
- Only for $n = 8$ the performances of GP are consistently worse wrt to the theoretical optimum.
A Posteriori Analysis – Implementation Properties, $n = 5$

Table: Power is in $nW$, area in $GE$, and latency in $ns$. $DPow$: dynamic power, $LPow$: cell leakage power

<table>
<thead>
<tr>
<th>Size</th>
<th>Rule</th>
<th>Keccak</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DPow</td>
<td>321.684 LPow: 299.725 Area: 17 Latency:0.14</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DPow</td>
<td>324.849 LPow: 308.418 Area: 17 Latency:0.14</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DPow</td>
<td>446.782 LPow: 479.33 Area: 24.06 Latency:0.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DPow</td>
<td>534.015 LPow: 493.528 Area: 26.67 Latency:0.17</td>
<td></td>
</tr>
</tbody>
</table>

▶ Results on par with the Keccak $\chi$ S-box
Example of Optimal CA S-box found by GP
Outline

Intro – AI in symmetric crypto

AI-based optimization methods in cryptography

AI-based computational models in cryptography

Conclusions
Conclusions and Perspectives

Summing up:

▶ Up to now, AI-based methods and models can help in solving certain specific design problems for symmetric ciphers.
▶ Many more open directions remain!

Open questions:

▶ take into account other primitives (e.g. permutation layers)
▶ Have a better understanding of which algorithm works best to evolve a Boolean function/S-box with certain properties (using e.g. fitness landscape analysis)
▶ Apply AI to other optimization problems in symmetric crypto (e.g. rotation constants selection)
http://keccak.noekeon.org/


