On the Applications of Evolutionary Algorithms to the Cybersecurity Domain

WEPO Workshop, November 30, 2021

Outline

1 Introduction

2 Boolean Functions

3 Substitution Boxes

4 Quantum Protocols

5 Physically Unclonable Functions

6 Hardware Trojans - EC

7 Fault Injection

Outline

1 Introduction

2 Boolean Functions

3 Substitution Boxes

4 Quantum Protocols

5 Physically Unclonable Functions

6 Hardware Trojans - EC

7 Fault Injection

Introduction

- We will notice that such artificial intelligence (AI) techniques are used more often in **attacks** than in **constructions**.
- More precisely, they are used more often in attacks by the crypto community.
- There are two main reasons for this:
  1. It is easier to validate that the attack works. Indeed, we require only a successful attack as proof. For constructions, it is difficult to capture all the notions of security when using data or fitness functions.
  2. Attacks are made after the constructions are done. So, there is the effect of timeliness. For constructions, one needs to use AI while designing the system, which is often not possible. Later, even if AI produces improved constructions, it is hard to change the already made design.

Outline

1 Introduction

2 Boolean Functions

3 Substitution Boxes

4 Quantum Protocols

5 Physically Unclonable Functions

6 Hardware Trojans - EC

7 Fault Injection

Boolean Functions

- The easiest problem to start.
- A natural mapping between the truth table representation of Boolean functions and representation of solutions in EC.
- Boolean functions are important cryptographic primitive and often used in stream ciphers as the source of nonlinearity.
- Boolean functions are commonly used in combiner or filter generators.

Boolean Functions

- Evolving Boolean functions is more interesting from the perspective of a difficult optimization problem, and not designing cryptographic primitive that will be used in ciphers.

Truth table input		Truth table output
x_1	x_0	y
0	0	0
0	1	1
1	0	1
1	1	0

Figure: Boolean function representation with truth table (two variables).

Boolean Functions

- Depending on the setting, we are interested in a number of properties (balancedness, nonlinearity, algebraic degree, correlation immunity, algebraic immunity), where some of those properties are conflicting.
- Search space size is 2^{2^n}.
- Representing solutions in the truth table form requires a string of bits of length 2^n.
- For smaller sizes, bitstring, integer, and floating-point also give good results.
- Currently, the best results, in general, are achieved with GP/CGP.
- Such results are comparable with those from algebraic constructions.

Boolean Functions

- Three main directions in the evolution of Boolean functions:
  1. Evolution of Boolean functions fulfilling a number of cryptographic properties (that can be used in combiner or filter generators). Additionally, with some special properties that are useful for masking countermeasures against side-channel attacks.
  2. Evolution of bent Boolean functions. Bent Boolean functions are maximally nonlinear but not balanced, and as such, not directly usable in crypto. Still, this represents an interesting benchmark problem.
  3. Evolution of algebraic constructions that are used to design Boolean functions.

Outline

1 Introduction

2 Boolean Functions

3 Substitution Boxes

4 Quantum Protocols

5 Physically Unclonable Functions

6 Hardware Trojans - EC

7 Fault Injection

S-boxes

- Natural extension from the Boolean function case.
- S-boxes (Substitution Boxes) are also called vectorial Boolean functions.
- Often used in block ciphers as a source of nonlinearity.
- However, this problem is much more difficult!
- S-box of dimension $n \times m$ has m output Boolean functions, but for several cryptographic properties we need to check all linear combinations of those functions (there are $2^n - 1$ linear combinations to consider).

Truth table input		Linear combinations			Walsh-Hadamard spectrum		
x_1	x_0	y_1	y_0	$y_1 \oplus y_0$	y_1	y_0	$y_1 \oplus y_0$
0	0	0	1	1	0	0	0
0	1	1	0	1	0	-4	0
1	0	1	1	0	0	0	-4
1	1	0	0	0	4	0	0

Figure: S-box with two inputs x and outputs y.

S-boxes

- For an S-box of size $n \times m$, the search space size equals 2^{m2^n}.
- Commonly (with EC), we explore cases where $n = m$, which means that for $n = m = 8$, the search space size equals $2^{2\,048}$.
- Common sizes to evolve with EC are from 3×3 to 8×8.
- Common solution representations are the same as for Boolean functions, plus permutation (which enforces bijectivity).
- Note, if using the tree representation, one actually evolves n trees.
- For smaller sizes, (up to 4×4) all solution representations work well.

S-boxes

- Similar to the Boolean function case, there are three main approaches to construct S-boxes: *i*) algebraic constructions, *i*) random search, and *iii*) heuristics.
- EC is commonly used to:
  1. Find bijective S-boxes with high nonlinearity (and low differential uniformity). Note that for such S-boxes, we know several algebraic constructions.
  2. To find S-boxes with additional properties. These commonly go into the direction of resilience against side-channel attacks.
  3. To find more efficient implementations of S-boxes (efficient in terms of area, power).

S-boxes

- The best results are obtained with tree representation and the cellular automata approach.
- CA representation was the first to obtain bijective S-boxes with optimal cryptographic properties for sizes up to 7×7 (not including 6×6 as there, no EC technique found the bijective S-box with the best possible differential uniformity).
- Already for size 8×8, EC results are far from those obtained with algebraic constructions (except if the initial population is seeded with good S-boxes).

S-boxes

- S-boxes are becoming less popular due to the rise of permutation-based cryptography, but they are still widely used.
- Naturally, most EC solutions are obtained much after the cipher design, so it is impractical to change the whole cipher simply to accommodate a new S-box.
- Interesting challenges for EC are *i*) obtain S-box of size 8×8 with the same properties as with algebraic constructions, and *ii*) evolve algebraic constructions of S-boxes (either primary or secondary).

Outline

1 Introduction

2 Boolean Functions

3 Substitution Boxes

4 Quantum Protocols

5 Physically Unclonable Functions

6 Hardware Trojans - EC

7 Fault Injection

Quantum Protocols

- Instead of operating on primitive or cipher level, EC can be used to evolve protocols also.
- As an example, it is possible to use EC to evolve novel quantum key distribution (QKD) protocols designed to counter attacks against the system in order to optimize the speed of secure communication.
- In essence, the goal is to evolve protocols as quantum circuits.
- Then, it is possible to define whether EC must work on the specific, user-defined template of a protocol or without explicit rules on how to access quantum communication channel.

Quantum Protocols

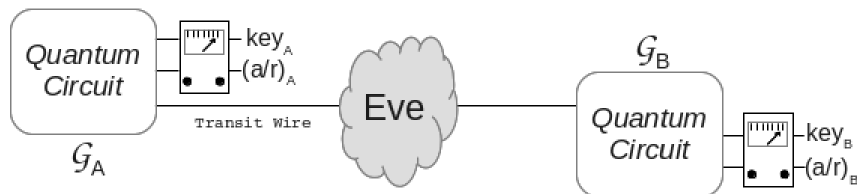

Figure: QKD protocol depicted as two circuits G_A and G_B. First, run circuit G_A, and then, the wire is measured, yielding a classical bit. Then, Eve is allowed to attack the transit line. Finally, G_B circuit is run, acting on the transit wire, and additional wires private to party B. Then, B's wire is measured.

Outline

1 Introduction

2 Boolean Functions

3 Substitution Boxes

4 Quantum Protocols

5 Physically Unclonable Functions

6 Hardware Trojans - EC

7 Fault Injection

Physically Unclonable Functions

- Physically Unclonable Functions (PUFs) are embedded or standalone devices used as a means to generate either a source of randomness or to obtain an instance-specific uniqueness for secure identification.
- This is achieved by relying on inherent uncontrollable manufacturing process variations, which results in each chip having a unique response.
- No two PUFs will give the same response when supplied with the same challenge.
- There exists no ideal PUF.
- Ideal PUF is unpredictable and without noise.
- Practical realizations depend on noise, aging, environmental variables, and process variations.

Physically Unclonable Functions

- Two types of PUFs: strong and weak.
- The difference with respect to the number of challenge-response pairs (CRPs) an attacker is allowed to obtain.
- The number of unique challenges c scales polynomially with the circuit area of a weak PUF.
- The number of unique challenges c scales exponentially with the circuit area of a strong PUF.

Physically Unclonable Functions

- Weak PUF has a limited number (typically, one or few) of responses to challenges.
- Strong PUFs have a large number of responses (with respect to different challenges).
- Strong PUFs have a virtually unlimited number of challenges c, but their CRPs are highly correlated.
- Given enough (often small amount) of CRPs, it is possible to build a predictive model of a strong PUF (in a way, we build a mathematical clone since it is not feasible to make analog physical clone).
- There exists no validated design of a strong PUF that is fully resilient against modeling attacks.

Physically Unclonable Functions

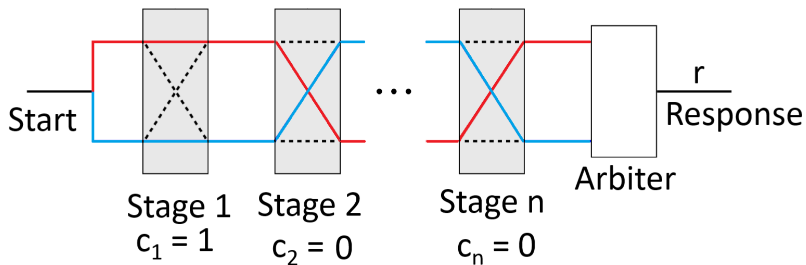

Figure: An example of a strong PUF - Arbiter PUF with n stages.

Physically Unclonable Functions

- Several techniques are commonly used to break strong PUFs.
- From ML domain, logistic regression, and from EC, evolution strategy.
- This domain is very interesting as AI provided results that were not possible to obtain with any other technique.
- What is more, even simple AI techniques can easily break strong PUFs.
- This also means there is not much development in the domain as attacks are easy to do, so no clear benefit of using more complex techniques, e.g., deep learning.

Outline

1 Introduction

2 Boolean Functions

3 Substitution Boxes

4 Quantum Protocols

5 Physically Unclonable Functions

6 Hardware Trojans - EC

7 Fault Injection

Hardware Trojans

- Hardware Trojans (HTHs) are malicious hardware components that intend to leak secret information or cause malfunctioning at run-time in the chip in which they are integrated.
- Over the last decade, HTHs have gained increasing attention.
- There are no HTHs in ICs reported in real-world applications yet, there are many examples of academic research results, both on injecting and detecting/preventing HTHs.
- HTHs can be inserted by untrusted foundries and actors at different stages in the design and development of FPGAs and ASICs (Application-Specific Integrated Circuits).

Hardware Trojans

- Conventional HTHs typically modify the functionality of target circuits at the register transfer level, net-list level, layout level, or dopant level to obtain secret information directly, to induce a fault for differential fault analysis, or to disable/degrade an embedded (pseudo) random number generator.
- HTH consists of two parts: a trigger and a payload.
- The trigger usually corresponds to a rare data input (sequence), while the payload is the activity that causes the data leakage or the malfunctioning when the HTH is triggered.
- HTHs are usually inserted in special places in the design that have low testability or high slack time.
- Testability is measured through two parameters: controllability and observability.

Hardware Trojans

- Compared to research concerned with the design of Hardware Trojans, considerably more results exist related to different Hardware Trojan detection mechanisms and countermeasures.
- Most research focuses on detecting Hardware Trojans inserted during manufacturing.
- In many cases, a golden model is used that is supposed to be Trojan free to serve as a reference.
- One important question is how to get to a Trojan-free golden model.

Outline

1 Introduction

2 Boolean Functions

3 Substitution Boxes

4 Quantum Protocols

5 Physically Unclonable Functions

6 Hardware Trojans - EC

7 Fault Injection

Fault Injection

- A fault injection (FI) attack is successful if, after exposing the device to a specially crafted external interference, it shows an unexpected behavior exploitable by the attacker.
- Insertion of signals has to be precisely tuned for the fault injection to succeed.
- Finding the correct parameters for a successful FI can be considered as a search problem where one aims to find, within a minimum time, the parameter configurations which result in a successful fault injection.
- The source of fault can be, e.g., voltage glitching, laser, electromagnetic radiation.
- Depending on the source of the fault, the search space of possible parameters changes significantly.
- In general, the search space is too big to conduct an exhaustive search.

Fault Injection

- Commonly, one defines several possible classes for classifying a single measurement:
    1. NORMAL: smart card behaves as expected, and the glitch is ignored
    2. RESET: smart card resets as a result of the glitch
    3. MUTE: smart card stops all communication as a result of the glitch
    4. CHANGING: the response is changing when repeating measurements.
    5. SUCCESS: smart card response is a specific, predetermined value that does not happen under normal operation

Fault Injection

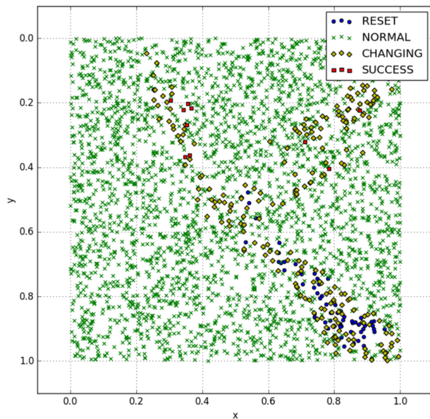

Figure: A depiction of search space for voltage glitching and two parameters.

Fault Injection

- Research domain with not many results from AI.
- Commonly used techniques are random search and grid search, so EC makes a strong alternative.
- The main issue is very expensive equipment to run fault injection campaigns.
- Mostly, EC is used, but recently, also deep learning found its place.
- Deep learning can be used to predict what a target would respond to a specific parameter combination.

Outline

1 Introduction

2 Boolean Functions

3 Substitution Boxes

4 Quantum Protocols

5 Physically Unclonable Functions

6 Hardware Trojans - EC

7 Fault Injection

Fuzzing

- Fuzzing (fuzz testing): automated software testing technique.
- Generating inputs and feeding them to the program being tested in the hope of evoking erroneous behavior or increasing code coverage.
- *Mutation-based fuzzing*: uses a dataset of test cases (a corpus):
  - selects a test case,
  - modifies it by applying *mutation operators*,
  - feeds it to the tested program.
- Example mutation operators: bit flip, random byte value, set byte to *interesting* value, insert byte, delete byte, ...
- Optimization of fuzzing: finding an appropriate sequence of mutation operators (*mutation scheduling*).

Fuzzing

- The fuzzing loop contains a deterministic stage (uses a predefined order of mutations) and randomized stage (random choice of mutations).
- The optimization is applied to the randomized stage, with the goal of finding the effective. *probability distribution* of mutation operators
- An example of *online* learning: optimization is performed concurrently with the process being optimized:
  - the mutation scheduler uses the current, solution (probability distribution) to select mutation operators,
  - the reported feedback is used to optimize the distribution,
  - the updated probability distribution is applied in the next iteration.
- Optimization is made under uncertainty: the fitness of a solution may only be estimated.

Fuzzing

- Examples including evolutionary optimization in mutation-based fuzzing:
- MOPT fuzzer: uses a variant of PSO to learn globally optimal mutation probability distribution
  - heavily dependent on choice of parameters and the tested program
  - may exhibit slow convergence due to multiple solutions in the population
- An approach with single-solution metaheuristic (e.g. $(\mu + \lambda)$ ES) could increase effectiveness:
  - focuses on increasing the speed of convergence,
  - more robust over different target programs.
- EC-based mutation schedulers outperform standardized fuzzing platforms.

Outline

1 Introduction

2 Boolean Functions

3 Substitution Boxes

4 Quantum Protocols

5 Physically Unclonable Functions

6 Hardware Trojans - EC

7 Fault Injection

Conclusions

- Logic locking.
- Side-channel analysis.
- Network intrusion.
- Malware detection.
- Adversarial examples.
- ...

Conclusions

- Up to now, EC proved to be successful in cryptography and cybersecurity.
- EC is used:
    1. When there exist no other, specialized approaches.
    2. To rapidly check whether some concept (e.g., formula) is correct.
    3. To assess the quality of some other method.
    4. To produce "good-enough" solutions.
    5. To produce novel and human-competitive solutions (solutions produced by EC that can rival the best solutions created by humans).

Conclusions

- We presented here only a handful of applications, there are more options.
- Even for each of the applications, there is a plethora of options still to try:
  1. New algorithms.
  2. Combinations of parameters.
  3. Representations.
  4. Fitness functions.
- The results obtained up to now are good, but there is still much room for improvement.

Questions?

Thanks for your attention!

stjepan@computer.org Q?