Evolutionary Computation and Cryptology

Stjepan Picek
Massachusetts Institute of Technology, CSAIL, USA
and
Cyber Security Research Group,
Delft University of Technology, The Netherlands
stjepan@computer.org

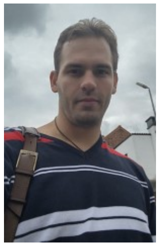
http://gecco-2017.sigevo.org/

GECCO '17 Companion, July 15-19, 2017, Berlin, Germany
ACM 978-1-4503-4939-0/17/07.
http://dx.doi.org/10.1145/3067695.3067725

1

Instructor

- ❖ Stjepan Picek is currently a postdoctoral researcher in ALFA group, CSAIL, MIT, USA. Prior to that, Stjepan was a postdoc researcher at KU Leuven, Belgium. Stjepan obtained his PhD in 2015 from Radboud University Nijmegen, The Netherlands and University of Zagreb, Croatia. His research topics include cryptology, evolutionary computation, and machine learning.



2

Agenda

- ❖ Introduction to Cryptology
- ❖ Evolutionary Computation
- ❖ Applications of EC to Cryptology
  - Boolean Functions
  - S-boxes
  - Addition Chains
  - Pseudorandom Number Generators
  - PUFs
  - Fault Injection
- ❖ Conclusions
- ❖ References

3

Introduction to Cryptology

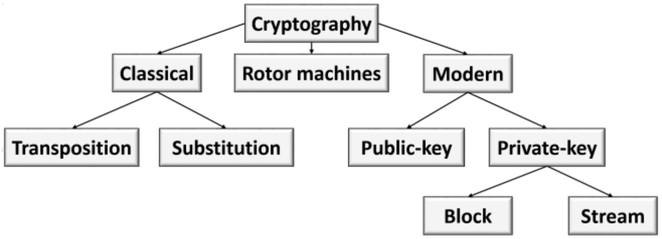
4

Introduction to Cryptology

- **Cryptology** (from Greek words kryptos which means hidden and logos which means word) is the scientific study of cryptography and cryptanalysis.
- We can trace the origins of cryptology in an art form to the ancient Egypt.
- **Cryptography** is a science (and art) of secret writing with the goal of hiding the meaning of a message. In modern cryptography, it is not only important to achieve confidentiality, but also authentication, non-repudiation and data integrity among other goals.
- **Cryptanalysis** is a science of analyzing ciphers in order to find weaknesses in them.

5

Introduction to Cryptology



Taxonomy

6

Classical Ciphers

- Transposition ciphers are such ciphers where the order of characters is shuffled around.
- Substitution ciphers are ciphers where each character in the alphabet is substituted with another character in the alphabet.
- Enigma machine is a mechanical rotor device that is comprised from several rotors that dynamically substitute the plaintext in accordance to the rotor position.
- Today, easy to cryptanalyze.
- Scytale, Caesar cipher, non-standard hieroglyphs, etc.

7

Modern Ciphers

- In 1940s Shannon published his paper on the design principles of block ciphers.
- Important milestones happened in 1970s.
- The design of the DES cipher, the introduction of public key cryptography.
- Modern cryptography has much more emphasize on definitions and proofs, although there are many primitives used today that do not have rigorous proofs.
- Informally, we distinguish classical from the modern cryptography on a basis that modern cryptography has a more scientific approach.

8

Basic Notions

- Sender is a person who is sending a message. The most famous sender in cryptography is Alice.
- Receiver is a person who is receiving a message. The most famous message receiver in cryptography is Bob.
- Adversary is a malicious entity whose aim is to prevent the users of a cryptosystem from achieving their goals. Popular names are Eve in the case of passive adversaries and Mallory when talking about active adversaries.
- Cryptographic primitive is a part of a cryptographic tool used to provide information security, i.e., a low-level cryptographic algorithm that is frequently used.

9

Basic Notions

- Cryptographic algorithm (cipher) is a mathematical function used for encryption, decryption, key establishment, authentication, etc.
- Plaintext *P* or message is the information that the sender wishes to transmit to the receiver.
- Ciphertext *C* is the result of an encryption performed on plaintext using a cryptographic algorithm.
- Encryption is a process of applying a transformation *E* to the plaintext *P*. After that transformation, only an authorized party should be able to read the message, i.e., *E(P) = C*.
- Decryption is a process of applying a transformation *D* to the ciphertext *C*, i.e., *D(C) = P*.

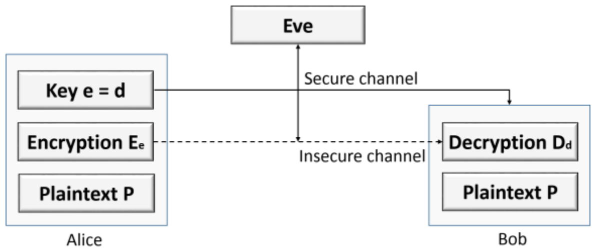
10

Symmetric-key Cryptography

- Also known as private key cryptography.
- Symmetric-key cryptography uses the same key to encrypt/decrypt or to compute/verify the data.
- Assume that Alice and Bob want to exchange some message and they want it to remain secret, i.e., that no one else can read it.
- They have only an insecure channel to communicate through. Alice could encrypt her message and send it encrypted over an insecure channel to Bob. If Bob has the same key as Alice, he can then decrypt and read the message.
- Eve cannot decrypt the message if she does not know the key.

11

Symmetric-key Cryptography



12

Block Ciphers

- Block ciphers operate on blocks of fixed length of data with an unvarying transformation that is specified by the key.
- Should be indistinguishable from a random permutation by an adversary not knowing the key.
- Claude Shannon stated that computationally secure cryptosystem should follow confusion and diffusion principles.
- Confusion – the ciphertext statistics should depend on the plaintext statistics in a manner too complicated to be exploited by the cryptanalyst.
- Diffusion - each digit of the plaintext and each digit of the secret key should influence many digits of the ciphertext.
- DES, AES, MARS, PRESENT, etc.

Stream Ciphers

- Should behave as pseudorandom number generators (PRNGs).
- Most of the stream encryption schemes encrypt message bits by adding encryption bits modulo two.
- Historically looking, linear feedback shift registers (LFSRs) were used in order to produce pseudorandom numbers.
- An LFSR is a shift register whose input bit is a linear function of its previous state. Those bit positions that affect the next state are called taps.
- To add the nonlinearity (and therefore improve the security) one option is to add some nonlinear element, where a Boolean function is a common choice.

Implementation Attacks

- All attacks that do not aim at the weaknesses of the algorithm itself, but on the implementations on cryptographic devices.
- Sources: power, sound, light, electromagnetic radiation, etc.
- Implementation attacks are among the most powerful known attacks against cryptographic devices.
- Common types of implementation attacks are side channel attacks and fault injection attacks.
- Side channel attacks are passive and non-invasive attacks.
- Fault injection attacks are active attacks since they enforce the target to work outside the nominal operation range.

Public-key Cryptography

- In symmetric-key cryptography, both parties need to know the key before the communication in order to establish the secure channel.
- However, the problem is how to exchange that key if there exists no secure channel.
- One option is to use public-key cryptography.
- Also called asymmetric cryptography.
- Here, there exist two keys: private and public key.
- To encrypt, one uses the public key, but to decrypt one needs to know the private key.

Public-key Cryptography

- Public-key cryptography relies on difficult problems in mathematics, like integer factorization, discrete logarithm problem, knapsack problem, etc.
- RSA, Diffie-Hellman, ECC,…
- For public-key cryptography, the are only a few papers where authors use evolutionary computation and the results are not spectacular.
- However, this is to be expected: it is much more difficult to design some cryptographic primitive here or to attack a system with evolutionary computation.

17

Evolutionary Computation

18

Evolutionary Computation

- Research area within computer science that draws inspiration from the process of natural evolution.
- Evolutionary algorithms are population based metaheuristic optimization methods that use biology inspired mechanisms like selection, crossover or survival of the fittest.
- Genetic Algorithm (GA), Holland, 1975.
- Tree based Genetic Programming (GP), Koza, 1992.
- Cartesian Genetic Programming (CGP), Miller, 1999.
- Evolution Strategy (ES), Rechenberg, Schwefel, 1970s.
- NSGA-II, Deb, 2002.

19

Applications of EC to Cryptology

20

Basics

- How to solve hard problems in cryptology?
- Problems need to be hard (to be worthwhile), but not too difficult (to be impossible to solve).
- Plenitude of problems and possible methods to solve them.
- Care needs to be taken that one does not select too difficult problems.
- Often, evolutionary computation is not used to provide the final solutions, but instead to help us to improve the results of some other technique.

GECCO BERLIN 2017

21

Evolutionary Computation Framework

- ECF is a C++ framework intended for application of any type of evolutionary computation.
- Developed by Evolutionary Computation group from Faculty of Electrical Engineering and Computing, Zagreb, Croatia:

  http://gp.zemris.fer.hr/

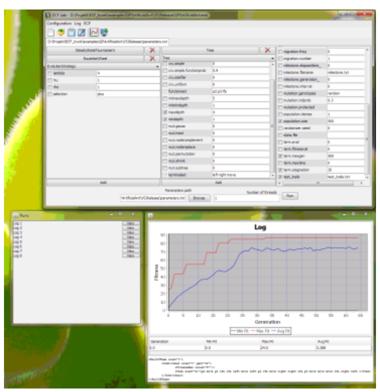
- Details about projects concerning evolutionary computation and cryptology:

  http://evocrypt.zemris.fer.hr/

GECCO BERLIN 2017

22

Evolutionary Computation Framework

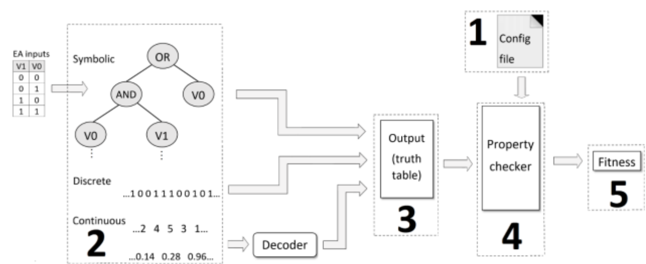


ECF GUI

GECCO BERLIN 2017

23

Evolutionary Computation Framework



1) Configure file with function size and cryptographic properties 2) Run evolutionary algorithm 3) Obtain truth table representation of a solution 4) Run Property checker 5) Use the checker's output as a metric of merit with values of desired properties.

GECCO BERLIN 2017

24

Boolean Functions

- The easiest problem to start.
- There exists 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.

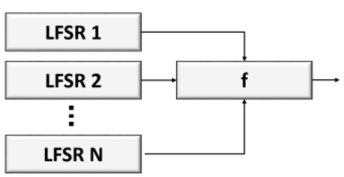
Truth table input		Truth table output
x_1	x_0	y
0	0	0
0	1	1
1	0	1
1	1	0

Boolean function with 2 inputs

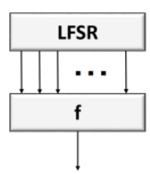
Boolean Functions

- To be used in cryptography, a Boolean function needs to fulfill a number of cryptographic properties.
- To be used in filter generators: balancedness, high nonlinearity, high algebraic degree, high algebraic immunity, high fast algebraic immunity.
- To be used in combiner generators additionally is required a good value of correlation immunity.
- To be used as a part of the side-channel attack countermeasure it needs to have low Hamming weight and high correlation immunity.
- To be of practical importance, it should have at least 13 inputs.

Boolean Functions



Combiner generator



Filter generator

Boolean Functions, Scenario 1

- Evolving Boolean functions that are to be used in combiner/filter generators.
- We are interested in a number of properties, where some of those properties are conflicting.
- Search space size is 2^{2^n}.
- Representing solutions in the truth table form requires string of bits of length 2^n.
- Already for a Boolean function with 8 inputs, the search space size is 2^{256}.

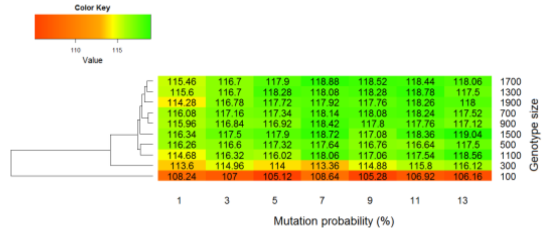
Boolean Functions, Scenario 1

- Fitness functions: single objective with the weight factors, multiple stage fitness function, multi-objective, many-objective.
- For Boolean functions up to 8 inputs, most of the EC techniques give good results.
- Currently, the best results are obtained with GP/CGP.
- The simplest problems seem to be either:
  - Evolving bent function (those that are not balanced, but with maximum nonlinearity)
  - Evolving balanced functions with high nonlinearity.

29

Boolean Functions, Scenario 1

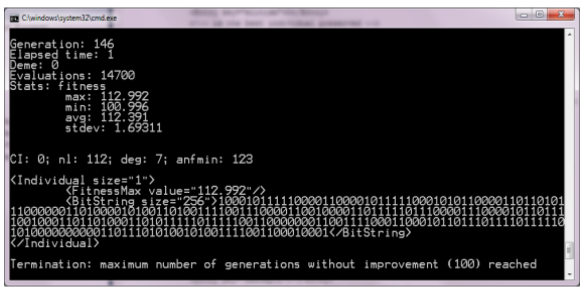
- Much larger role of genotype than the choice of fitness function.



Average values, CGP, bent Boolean functions with 8 inputs

30

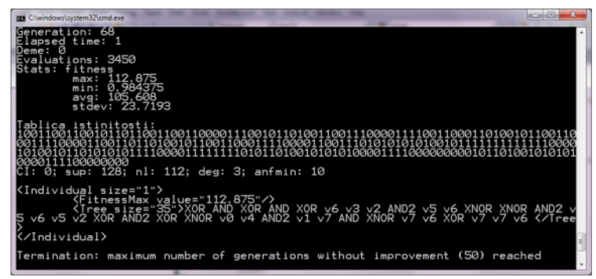
Boolean Functions, Scenario 1



GA, bitstring representation Boolean function with 8 inputs

31

Boolean Functions, Scenario 1



GP, Boolean function with 8 inputs

32

Boolean Functions, Scenario 2

- Evolve Boolean functions with as small as possible Hamming weight and high correlation immunity in order to reduce the masking cost.
- Masking consists in changing randomly the representation of the key to deceive the attacker.
- Example: if each bit k_i, $1 < i < n$ of a key k is masked with a random bit m_i, then an attacker could probe k_i *XOR* m_i.
- Provided m_i is uniformly distributed, the knowledge of k_i *XOR* m_i does not disclose any information on bit k_i.
- Since most of the algebraic constructions aim to find balanced Boolean functions, they are not appropriate for this problem.

Boolean Functions, Scenario 2

- Masking can be summarized as the problem of finding Boolean functions whose support is the masks' set, with the two following constraints:
  - small Hamming weight, for implementation reasons, and
  - high correlation immunity t to resist an attacker with multiple ($< t$) probes.
- There is a trade-off which motivates the research for low Hamming weight high correlation immunity Boolean functions.
- Interesting problem since we know the best possible values, but we do not know actual functions reaching those values.

Boolean Functions, Scenario 2

- Up to recently, there were several values of practical interest unknown.
- Attempts with SAT solvers did not resulted in success even after more than one month of calculation.
- For CGP and GP, this problem seems to be trivial.
- Optimal results sometimes achieved even in less than 1 hour.
- However, there are combinations of parameters as well as function sizes that seem more difficult for EC.

Boolean Functions, Scenario 2

t \ n	11	12	13	14	15	16
2	16	16	32	64	128	256
3	32	32	32	64	128	256
4	**128**	256	256	256	2 048	4 096
5	**256**	**256**	512	1 024	2 048	4 096
6	512	**1 024**	1 024	2 048	4 096	4 096
7	1 024	1 024	**2 048**	4 096	8 192	8 192
8	1 024	2 048	4 096	8 192	8 192	16 384
9	1 024	2 048	4 096	8 192	16 384	16 384
10	1 024	2 048	4 096	8 192	16 384	32 768
11		2 048	4 096	8 192	16 384	32 768
12			4 096	8 192	16 3840	32 768
13				8 192	16 384	32 768
14					16 384	32 768
15						32 768

Solutions with GP and CGP

Boolean Functions, Scenario 3

- Previous results show that EC can be used to evolve Boolean functions of various sizes and properties.
- However, it is to be expected that after some size, the results will become worse and the evaluation process long.
- For instance, if we consider the algebraic immunity and fast algebraic immunity properties. To calculate those two properties can easily take several hours for a Boolean functions with e.g. 16 inputs.
- Therefore, at least for now, those properties were never included in the evaluation process for larger sizes of Boolean functions.

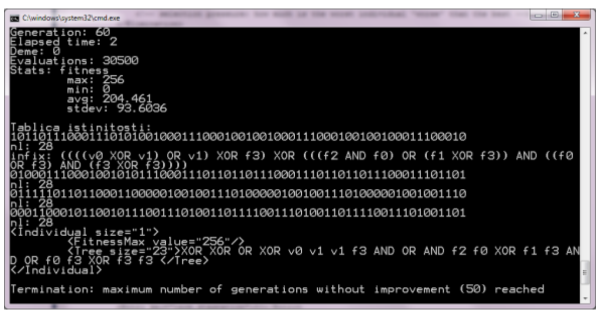
37

Boolean Functions, Scenario 3

- We already discussed there are several techniques how to generate Boolean functions.
- The question is can we combine several techniques.
- For instance, could we use evolutionary computation to evolve algebraic constructions?
- If yes, then we need just to show that our construction results in Boolean functions with good properties and that it holds for any size of Boolean functions.
- We evolve secondary algebraic constructions that result in bent Boolean functions.
- 2016 GECCO Humies finalist.

38

Boolean Functions, Scenario 3



GP secondary construction

39

Boolean Functions, Perspectives

- Possible challenges:
  - Finding balanced Boolean function with 8 inputs that have nonlinearity 118.
  - Use EC to evolve primary algebraic constructions.
  - Evolve Boolean functions to be used in combiner/filter generators where parameters are also algebraic immunity and fast algebraic immunity.
  - Use different, previously not investigated unique representations of Boolean functions.
  - Investigate many-objective optimization.
  - Quaternary Boolean functions.

40

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 *nxm* has *m* output Boolean functions, but for the most of the properties we need to check all linear combinations of those functions.

S-boxes

Truth table input		Truth table output		LUT input	LUT output
x_1	x_0	y_1	y_0	x	y
0	0	0	1	0	1
0	1	1	0	1	2
1	0	1	1	2	3
1	1	0	0	3	0

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

2x2 S-box

S-boxes

- For an S-box with n inputs and m outputs, there are in total 2^{m2^n} S-boxes.
- Some realistic search space sizes when n=m:

n	4	5	6	7	8
#	2^{64}	2^{160}	2^{384}	2^{896}	$2^{2\,048}$

- Several options to represent solutions.
- As with Boolean functions, there are three design options: algebraic constructions, random search, and heuristics.

S-boxes, Scenario 1

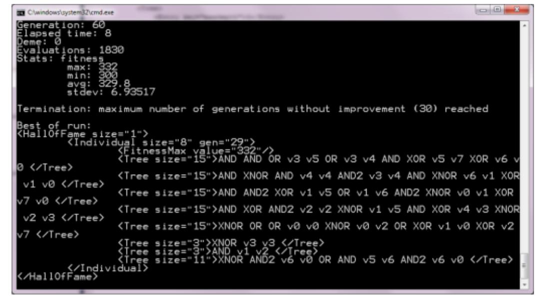
- When representing S-boxes with their truth tables (i.e., bitstring representation as with Boolean functions) the problem is very difficult.
- Already balancedness property requires that all columns of an S-box are balanced (have the same number of zeros and ones), but also all linear combinations needs to be balanced.
- Still, this approach works for sizes ~4x4 where there are 15 linear combinations we need to consider.
- However, for larger sizes, it is almost impossible to obtain even balanced solution with bitstring representation.
- Therefore, we do not consider such representation anymore.

S-boxes, Scenario 1

- It is possible to use CGP and GP with the permutation encoding:

```
Algorithm 1 Translate to permutation encoding.
Require: i = 0, m = input_size, n = output_size
  for all values i < 2^m do
    balanced[i] = i
    for j = n-1; j != 0; j-- do
      evolved[i] = evolved[i] + truth_table[i][j]*(2^j)
    end for
  end for
  SORT balanced[] array using evolved[] array as key
  for all values i < 2^m do
    for j = n-1; j != 0; j-- do
      truth_table[i][j] = (balanced[i] * 2^j) & 0x01
    end for
  end for
```


S-boxes, Scenario 1



GP solution of an 8x8 S-box

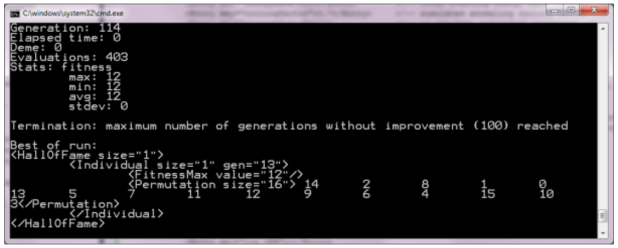
S-boxes, Scenario 2

- Represent S-boxes as permutations, i.e., all values between 0 and $2^n - 1$ (where n is the dimension of the S-box).
- Then the S-box is always bijective and we do not need to worry about the balancedness property.
- Similar as with Boolean functions, there are many properties of interest when evolving S-boxes: high nonlinearity, low differential uniformity, high algebraic degree, etc.
- For dimensions up to 4x4, permutation encoding gives optimal results (bijective solutions with maximal nonlinearity and minimal differential uniformity).
- For 8x8, algebraic construction gives nonlinearity of 112 and differential uniformity of 4.

S-boxes, Scenario 2

- Random search results in nonlinearity up to 98 and nonlinearity down to 10.
- Heuristics - up to 104 nonlinearity, differential uniformity 8.
- The question is then whether there is any sense to use heuristics if such methods cannot compete with algebraic constructions.
- It turns out there are properties that algebraic constructions do not consider. Properties related with the side-channel resistance often have poor values if S-boxes are constructed with algebraic constructions.
- Evolve S-boxes with good side channel resistance while keeping other properties optimal.

S-boxes, Scenario 2



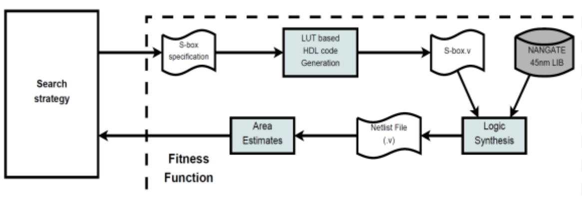
```
C:\windows\system32\cmd.exe
Generation: 114
Elapsed time: 0
Deme: 0
Evaluations: 403
Stats: fitness
        max: 12
        min: 12
        avg: 12
        stdev: 0

Termination: maximum number of generations without improvement (100) reached

Best of run:
<HallOfFame size="1">
        <Individual size="1" gen="13">
                <FitnessMax value="12"/>
                <Permutation size="16"> 14      2       8       1       0
13      5       7       11      12      9       6       4       15      10
3</Permutation>
        </Individual>
</HallOfFame>
```

Permutation encoding of an 4x4 S-box

S-boxes, Scenario 3

- Besides the properties related with the side-channel attacks, we are also interested in implementation properties like power, area, and latency.
- Again, algebraic constructions do not consider such properties but we can evolve S-boxes with good cryptographic properties that are hardware-friendly.
- Naturally, there exist the same problem as before: we do not want that cryptographic properties deteriorate too much.
- In this scenario, we require that our evolution framework can communicate with the framework that does the implementation properties analysis.

S-boxes, Scenario 3



Evaluation setup when evolving S-boxes with good implementation properties

S-boxes, Scenario 3

- EC cannot handle larger S-box sizes so we modify our approach.
- We evolve affine transformations of an S-box.
- We change implementation properties, while keeping most of the cryptographic properties intact:

$$S_a(x) = B(S_b(A(x) \text{ XOR } a)) \text{ XOR } b.$$

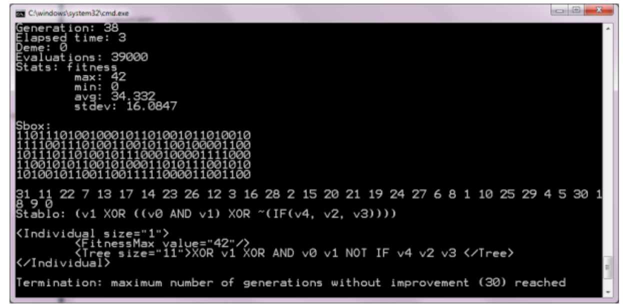
- A and B are invertible nxn matrices in GF(2) and a and b are constants.

S-boxes, Scenario 4

- Evolve S-boxes in a form of cellular automata (CA) rules.
- Such representation is also used in practice (Keccak cipher).
- The best results with EC up to now!

S-box size	T_max	GP			Ours		Related	
		Max	Avg	SD	N_F	δ_F	N_F	δ_F
4×4	16	**16**	16	0	4	4	4	4
5×5	42	**42**	41.73	1.01	**12**	**2**	10	4
6×6	86	**84**	80.47	4.72	**24**	**4**	22	6
7×7	182	**182**	155.07	8.86	**56**	**2**	48	6
8×8	364	318	281.87	13.86	94	20	**104**	**8**

S-boxes, Scenario 4



Evolved CA rule for the 5x5 S-box

S-boxes, Perspectives

- Possible challenges:
  - Evolve S-box of size 8x8 that has nonlinearity 112.
  - Use new representations of solutions.
  - Improve the efficiency of EC with the bitstring representation.
  - Consider S-box representations in a form of equations.
  - Find general rules for CA and S-boxes.
  - S-boxes where the number of inputs and outputs is not the same.

Addition Chains

- Modular exponentiation: find the (unique) integer $B \in [1, \ldots, p-1]$ such that:

$$B = A^c \bmod p.$$

- Several ways to calculate this.
- The simplest is to naïve multiply A c times.
- Addition chain: a sequence of positive integers where each value is a sum of two values occurring in the sequence.
- The length of an addition chain determines the number of multiplications required for exponentiation.

Addition Chains

- The aim is to find the shortest addition chain for a given exponent *c*.
- Binary method: write 60 in binary: 111100; replace “1” with “DA” and “0” with “D”; cross out the first “DA” on the left; “DADADADD”, calculate:

  1 → 2 → 3 → 6 → 7 → 14 → 15 → 30 → 60.
- Addition chain (7 operations):

A^1; A^2 = A^1 * A^1; A^4 = A^2 * A^2; A^6 = A^4 * A^2;

A^12 = A^6 *A^6;A^24 = A^12 * A^12; A^30 = A^24 * A^6;

A^60 = A^30 * A^30.

Addition Chains

- The problem of finding the shortest addition chain for a given exponent is of great relevance in cryptography.
- However, the problem is believed to be NP-hard.
- There is no single algorithm that can be used for any exponent.
- Still the best solutions are often obtained by pen and paper method.
- Huge numbers so exhaustive search is impossible.
- Heuristics should be able to help.

Addition Chains

- Types of steps in the addition chain:
  - Doubling step; when *j* = *k* = *i* - *1*. This step always gives the maximal possible value at the position *i*.
  - Star step: when *j* but not necessarily *k* equals *i* – *1*.
  - Small step: when $log_2(a_i) = log_2(a_{i-1})$.
  - Standard step: when $a_i = a_j + a_k$ where *i* > *j* > *k*.
- A star chain is a chain that involves only star operations.

Addition Chains

Algorithm 1 Crossover operator.

Require: Exponent $exp > 0$, *Parent addition chains* P_1, P_2

$rand = random(3, exp - 1)$
for all i such that $0 \leq i \leq rand$ **do**
  $e_i = P_{1_i}$
end for
for all i such that $rand \leq i + 1 \leq n$ **do**
  $FindLowestPair(P_2, i, pair_1, pair_2)$
  $e_i = e_{pair_1} + e_{pair_2}$
end for
$RepairChain(e, exp)$
return $e = e_0, e_1, ..., e_n$

Crossover operator for addition chains, needs to include repair mechanism

Addition Chains

Algorithm 2 Mutation operator.

```
Require: Exponent exp > 0, e = e_0, e_1, ..., e_n
  rand = random(2, exp − 1)
  rand_2 = random(0, 1)
  if rand_2 then
    e_rand = e_{rand−1} + e_{rand−2}
  else
    rand_3 = random(2, rand − 1)
    e_rand = e_{rand−1} + e_{rand_3}
  end if
  RepairChain(e, exp)
  return e = e_0, e_1, ..., e_n
```

Mutation operator for addition chains, needs to include repair mechanism

Addition Chains

r	c(r)	Binary	Optimized window	GA Min	GA Avg	GA Stdev
26	1 176 431	33	31	26	26.18	1.27
27	2 211 837	36	32	27	27.18	1.68
28	4 169 527	37	32	28	28.18	0.38
29	7 624 319	36	33	29	30.16	0.71
30	14 143 037	38	34	30	30.92	0.6
31	25 450 463	38	35	31	32.62	0.66
32	46 444 543	42	36	32	33.5	0.54
33	89 209 343	42	38	33	34.46	0.81
34	155 691 199	42	39	34	35.44	1.03
35	298 695 487	46	41	35	35.67	0.74
36	550 040 063	45	41	36	37.96	0.83
37	994 660 991	46	42	37	38.76	1.47
38	1 886 023 151	48	42	38	40.28	1.21
39	3 502 562 143	48	43	39	41.36	1.19
40	6 490 123 999	52	45	41	41.77	0.63

Results for a number of different values

Addition Chains

Exponent	$log_2(n)$	$\nu(n)$	Binary	Window	Optimized win.	GA Min	GA Avg	GA Stdev
$2^{37}-3$	36	35	71	57	51	43	45.32	0.99
$2^{47}-3$	46	45	91	69	63	54	56.25	1.11
$2^{57}-3$	56	55	111	82	76	64	64.9	0.87
$2^{67}-3$	66	65	131	94	88	73	73.2	0.43
$2^{77}-3$	76	75	151	107	101	85	85.4	0.51
$2^{87}-3$	86	85	171	119	113	97	104.3	3.56
$2^{97}-3$	96	95	191	132	126	106	107.2	0.91
$2^{107}-3$	106	105	211	144	138	115	115.71	0.75
$2^{117}-3$	116	115	231	157	151	126	126.6	0.89
$2^{127}-3$	126	125	251	169	163	136	136.8	0.83

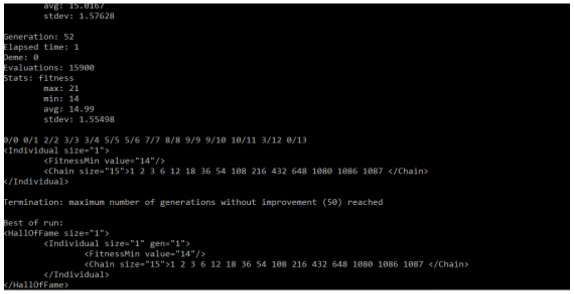
Results for a number of different values

Addition Chains

- For most of the values we find the optimal one (or what is the current best).
- Out of all tested numbers, only $2^{127}-3$ has practical importance.
- We find chain of 136 steps, also done by expert by hand.
- Human-competitive?
- We believe so, on average we need 10 minutes, pen and paper requires a lot of experience and will last longer.
- More realistic numbers are $2^{255}-21$ and

$2^{252}-27742317777372353535851937790883648491$.

Addition Chains



```
        avg: 15.0167
        stdev: 1.57628

Generation: 52
Elapsed time: 1
Deme: 0
Evaluations: 15900
Stats: fitness
        max: 21
        min: 14
        avg: 14.99
        stdev: 1.55498

0/0 0/1 2/2 3/3 3/4 5/5 5/6 7/7 8/8 9/9 9/10 10/11 3/12 0/13
<Individual size="1">
        <FitnessMin value="14"/>
        <Chain size="15">1 2 3 6 12 18 36 54 108 216 432 648 1080 1086 1087 </Chain>
</Individual>

Termination: maximum number of generations without improvement (50) reached

Best of run:
<HallOfFame size="1">
        <Individual size="1" gen="1">
                <FitnessMin value="14"/>
                <Chain size="15">1 2 3 6 12 18 36 54 108 216 432 648 1080 1086 1087 </Chain>
        </Individual>
</HallOfFame>
```

Example of an evolved addition chain

Addition Chains, Perspectives

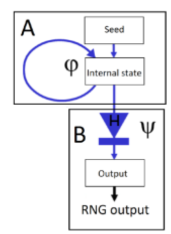
- Possible challenges:
  - Improve the speed of the algorithm.
  - Look for optimal chains for even larger numbers.
  - Differentiate between multiplication and squaring steps.
  - Analyze the structure of numbers with regards to the EC performance.
  - Support special structures of numbers.
  - Explore different types of chains.

Pseudorandom Number Generators

- In cryptography, random number generators (RNGs) play an important role.
- Most of the time, we need true random number generators (TRNGs), but still there are applications where pseudorandom number generators (PRNGs) are enough.
- TRNG is a device for which the output values depend on some unpredictable source that produces entropy.
- PRNGs represent mechanisms that produce random numbers by performing a deterministic algorithm on a randomly selected seed.
- One example is masking for the side channel resistance.

Pseudorandom Number Generators

- Find extremely fast and small PRNGs that pass all NIST statistical tests.
- Use GP and CGP to evolve PRNGs.



PRNG model

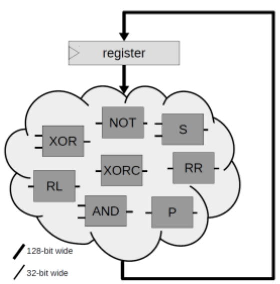
Pseudorandom Number Generators

- Evolve PRNGs that have *n* inputs and 1 output (GP) or *m* outputs (CGP).
- All variables are 32-bit integer values.
- Function set are function that are fast and small when implemented in hardware (shift, rotate, permute, and logical operations XOR, NOT, AND).
- Here, obvious advantage of CGP over GP is that GP needs to iterate *m* times to produce the same size of the output as CGP produces in a single iteration.
- Fitness function needs to be simple, yet powerful enough to drive our search.

Pseudorandom Number Generators

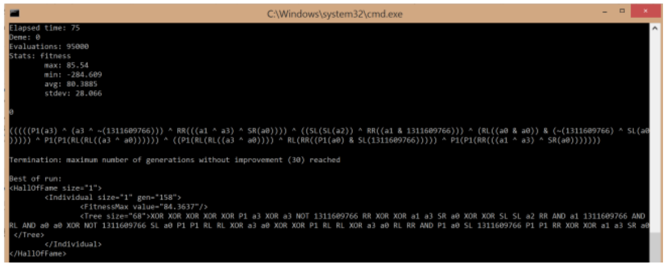
- We use approximate entropy test from the NIST statistical test suite as a fitness function.
- After the evolution process is over, our parser automatically takes the best individual and outputs it as a C source code.
- That source code is then used to produce 10 million bits that are then evaluated with the NIST statistical suite.
- We cannot use whole test suite in the evolution since it would be too slow.
- Our current fitness function consists of 130 evaluations of the approximate entropy function.

Pseudorandom Number Generators



Structure of evolved PRNGs

Pseudorandom Number Generators



Example of evolved PRNG

Pseudorandom Number Generators

```
uint GP(uint a0, uint a1, uint a2, uint a3) {
    uint x0 = (const >> 1) | (const << 31);
    uint x1 = a2 & a3;
    uint x2 = x1 ^ x0;
    uint x3 = p1(a3);
    uint x4 = p1(x3);
    uint x5 = x4 ^ x2;
    uint x6 = (a0 << 1) | (a0 >> 31);
    uint x7 = p1(x6);
    uint x8 = (x7 << 1) | (x7 >> 31);
    uint x9 = x8 ^ x5;
    uint x10 = (a0 >> 1) | (a0 << 31);
    uint x11 = a0 ^ a3;
    uint x12 = x11 ^ x10;
    uint x13 = p1(x12);
    uint x14 = a3 & a1;
    uint x15 = (x14 >> 1) | (x14 << 31);
    uint x16 = (a0 >> 1) | (a0 << 31);
    uint x17 = a3 ^ x16;
    uint x18 = x17 ^ x15;
    uint x19 = x18 ^ x13;
    uint x20 = x19 ^ x9;
    return x20;
}
```

GP solution

Pseudorandom Number Generators

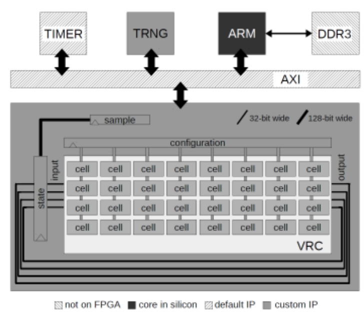
```
void CGP(uint x0,uint x1,uint x2,uint x3,
uint* z0,uint* z1,uint* z2,uint* z3) {
    uint y4 = x0 & x1;
    uint y5 = x2 ^ x3;
    uint y6 = (y5 >> 1) | (y5 << 31);
    uint y7 = p1(y6);
    uint y8 = x3 ^ y7;
    uint y9 = p1(y8);
    uint y10 = y6 ^ y9;
    uint y11 = (y9 << 1) | (y9 >> 31);
    uint y12 = const;
    uint y13 = p1(y10);
    uint y14 = y12 ^ y11;
    uint y15 = y12 ^ y13;
    uint y16 = (y15 >> 1) | (y15 << 31);
    uint y17 = y10 ^ y16;
    uint y18 = p1(y17);
    uint y19 = y18 >> 1;
    uint y20 = y18 ^ y4;
    uint y21 = p1(y20);
    uint y22 = y18 ^ y21;
    uint y23 = p1(y18);
    uint y24 = y19 ^ y18;
    uint y25 = y23 ^ y19;
    uint y26 = y22 ^ y14;
    *z0 = y18;
    *z1 = y25;
    *z2 = y26;
    *z3 = y24;
}
```

CGP solution

Pseudorandom Number Generators

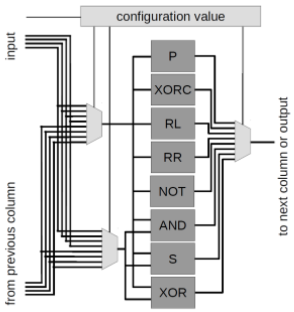
- The same technique can be used to produce PRNGs on-the-fly.
- Then, we can use evolvable hardware that constantly updates the PRNG part.
- In order to ensure that our designs always use all terminals, we penalize solutions that do not have all inputs.
- Maximal throughput on ASIC 117 Gb/s and for FPGA 66 Gb/s.
- Here, GP and CGP are used to evolve only the update functions, but EC can be also used to evolve the non-invertible function.

Pseudorandom Number Generators



Evolvable hardware setting

Pseudorandom Number Generators



Virtual reconfigurable circuit cell

Pseudorandom Number Generators, Perspectives

- Possible challenges:
  - Improve the fitness function and consequently the evaluation process.
  - Add to the fitness function also consideration about the size and speed of specific functions (platform dependent).
  - Experiment with different sizes of the update function as well as different terminal sets.
  - Improve the efficiency of the evolvable hardware scenario.

PUFs

- 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.
- Optimization techniques can be used to find a model ("clone") of a PUF by modeling the delay vector of an actual PUF in as few measurements as possible.

PUFs

- Arbiter PUF consists of one or more chains of two 2-bit multiplexers that have identical layouts.
- Each multiplexer pair is denoted a stage, with n stages in a single chain.
- There is a single input signal that is introduced to the first stage to both bottom and top multiplexer in the pair (red and blue).
- The chain is fed a control signal of n bits called a challenge (bits c_1 to c_n), where each bit determines whether the two input signals in that stage would be switched (crossed over) or not.

PUFs

- The response of a PUF is determined by the delay difference between the top and bottom input signal, which is in turn the sum of delay differences of the individual stages.
- To efficiently model a PUF, one usually tries to determine the delay vector w=$(w_1, \ldots, w_{n+1})$.
- The delay difference ΔD at the end of a chain is

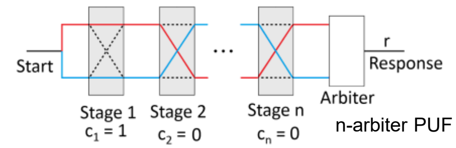
$$\Delta D = w^T \phi$$

- The feature vector ϕ is derived from the challenge vector as

$$\phi_i = \prod_{l=1}^{n} (-1)^{c_l}, \text{ for } 1 \le i \le n \text{ and with } \phi_{n+1} = 1.$$

- The final response is equal to 1 if ΔD < 0 and 0 otherwise.

PUFs



n-arbiter PUF

chains×stages/challenges	4x64/128	6x64/256	4x64/1 000
GA	0	1	221
GA/hybrid	0	1	102
CMA-ES	0	0	112
ClonAlg	0	19	218
OptIA	0	11	179

Examples of solutions

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.
- 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 search space is typically too large to perform an exhaustive search.
- Use heuristics to find search space parameters that lead to successful attack.

Fault Injection

- Voltage switching, three parameters: glitch length, glitch voltage, and glitch offset.
- Two scenarios:
  - Finding faults in a minimal number of measurements.
  - Characterizing the parameter space, again in a minimal number of measurements.
- FI testing equipment can output only verdict classes that correspond to successful measurements.
- Attacking the PIN mechanism.

Fault Injection

```
for (i = 0; i < 4; i++)
{
    if (pin[i] == input[i])
        ok_digits++;
}
if (ok_digits == 4)  //LOCATION FOR ATTACK
    respond_code(0x00, SW_NO_ERROR_msb, SW_NO_ERROR_lsb);
else
    respond_code(0x00, 0x69, 0x85);
```

PIN mechanism

Fault Injection

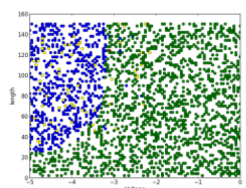
❖ Possible classes for classifying a single measurement:
- NORMAL: smart card behaves as expected and the glitch is ignored
- RESET: smart card resets as a result of the glitch
- MUTE: smart card stops all communication as a result of the glitch
- INCONCLUSIVE: smart card responds in a way that cannot be classified in any other class
- SUCCESS: smart card response is a specific, predetermined value that does not happen under normal operation

Fault Injection

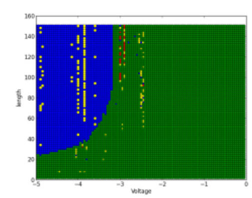
Input: crx_count = 0, mut_count = 0
repeat
  select first parent
  if first parent of SUCCESS class **then**
    try to find matching second parent
  else
    try to find second parent of different class
  end if
  perform crossover (depending on parent classes)
  copy child to new generation
  crx_count = crx_count + 1
until $p_c * N <$ crx_count
repeat
  select random individuals for tournament
  copy best of tournament to new generation
  mut_count = mut_count + 1
until $(1 - p_c) * N <$ mut_count
perform mutation on new generation with probability p_m
evaluate population

Custom GA for fault injection

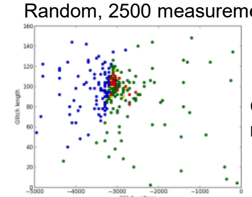
Fault Injection



Random, 2500 measurements

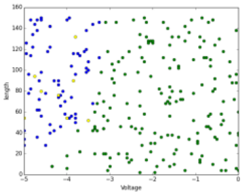


Exhaustive, 7500 measurements

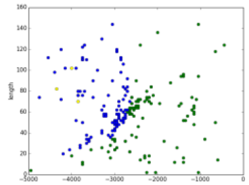


GA + LS, 250 measurements

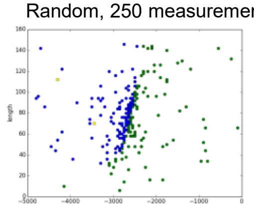
Fault Injection



Random, 250 measurements



GA, 250 measurements



GA+LS, 250 measurements

89

Fault Injection

- Possible challenges:
  - Working with more relevant parameters.
  - Attacking cards with countermeasures.
  - Switching to other sources of attacks.
  - Making the search algorithm more powerful.
  - Laser and electromagnetic radiation attacks.

90

Conclusions

91

Final Remarks

- All the examples presented here are available from SVN repository:

  http://evocrypt.zemris.fer.hr/

- In all the experiments we use Evolutionary Computation Framework (ECF) that can be downloaded from:

  http://ecf.zemris.fer.hr/

For updated version of slides as well as for the further references, please check:

http://www.evocrypt.com/

92

Security Applications

- Stepping outside of the cryptology area and considering security area there are many more interesting problems:
  - Malware detection.
  - Intrusion detection.
  - Automatic code improvement.
  - Spam detection.
- For EC applications in security, check the tutorial "Evolutionary Computation in Network Management and Security" by Nur Zincir-Heywood and Gunes Kayacik.

93

Machine Learning and SCA

- Side-channel attacks (SCA) represent extremely powerful category of attacks on cryptographic devices with profiled side-channel attacks in a prominent place as the most powerful among them.
- Within the profiling phase the attacker estimates leakage models for targeted intermediate computations, which are then exploited to extract secret information from the device in the actual attack phase.
- Classification and regression problems.
- Different devices, algorithms, number of classes, number of features, levels of noise, datasets, etc.
- Machine learning, deep learning, EC, etc.

94

Perspectives

- We also need to step outside the EC area and consider other heuristic techniques.
- Even for each of the applications, there is a plethora of options still to try:
  - New algorithms.
  - Representations.
  - Fitness functions.
  - Combinations of parameters.
- The results obtained up to now are good, but there is still much room for improvement.

95

Conclusions

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

96

Conclusions

- Heuristic methods are not a magic solvers.
- They require knowledge and experience if to be used correctly.
- Nice problems, both from the practical perspective, but also as benchmarks – see talk on crypto problems for benchmarking – CryptoBench.
- If there are others, specialized algorithms, EC rarely can beat them.

97

Conclusions

- Without proper collaboration, for EC community cryptology problems are just something to be solved but without adequate understanding.
- For cryptographic community, EC techniques are just a tool to be used.
- Without good understanding the problem and the tool to be used, it is hard to expect nice results.
- Thank you for your attention.

Questions?

98

Acknowledgements

- This work has been supported in part by Croatian Science Foundation under the project IP-2014-09-4882.
- The author would like to thank prof. Domagoj Jakobovic for his help with the preparation of presentation.

99

References

- J. Katz and Y. Lindell. Introduction to Modern Cryptography. Chapman and Hall/CRC, Boca Raton, 2nd edition, 2015.
- L. R. Knudsen and M. Robshaw. The Block Cipher Companion. Information Security and Cryptography. Springer, 2011.
- A. J. Menezes, S. A. Vanstone, and P. C. V. Oorschot. Handbook of Applied Cryptography. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 1996.
- B. Schneier. Applied cryptography (2nd ed.): protocols, algorithms, and source code in C. John Wiley and Sons, Inc., New York, NY, USA, 1995.
- J. H. Holland. Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence. The MIT Press, Cambridge, USA, 1992.
- J. R. Koza. Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge, MA, USA, 1992.
- J. F. Miller, editor. Cartesian Genetic Programming. Natural Computing Series. Springer Berlin Heidelberg, 2011.
- H.-G. Beyer and H.-P. Schwefel. Evolution Strategies A Comprehensive Introduction. Natural Computing, 1(1):3–52, May 2002.
- A. E. Eiben and J. E. Smith. Introduction to Evolutionary Computing. Springer-Verlag, Berlin Heidelberg New York, USA, 2003.

100

References

- J. F. Miller. An Empirical Study of the Efficiency of Learning Boolean Functions using a Cartesian Genetic Programming Approach. Genetic and Evolutionary Computation Conference (GECCO) 1999, pp. 1135–1142.
- L. D. Burnett. Heuristic Optimization of Boolean Functions and Substitution Boxes for Cryptography, Ph.D. thesis, Queensland University of Technology (2005).
- C. Carlet. Boolean Functions for Cryptography and Error Correcting Codes. Boolean Models and Methods in Mathematics, Computer Science, and Engineering, 1st Edition, Cambridge University Press, New York, USA, 2010, pp. 257–397.
- C. Carlet and S. Guilley. Correlation-immune Boolean functions for easing counter measures to side-channel attacks. Algebraic Curves and Finite Fields. Cryptography and Other Applications., Berlin, Boston: De Gruyter., 2014, pp. 41–70.
- W. Millan, J. Fuller, and E. Dawson. New concepts in evolutionary search for Boolean functions in cryptology, Computational Intelligence 20 (3) (2004) pp. 463–474.
- S. Picek, D. Jakobovic, and M. Golub. Evolving Cryptographically Sound Boolean Functions. Genetic and Evolutionary Computation Conference (GECCO) Companion 2013, pp. 191–192.
- S. Picek, L. Batina, and D. Jakobovic. Evolving DPA-Resistant Boolean Functions. PPSN XIII, Lecture Notes in Computer Science, Springer International Publishing, 2014, pp. 812–821.
- W. Millan, A. Clark, and E. Dawson. An Effective Genetic Algorithm for Finding Highly Nonlinear Boolean Functions. ICICS '97, pp.149–158.

References

- A. J. Clark. Optimisation heuristics for cryptology, Ph.D. thesis, Queensland University of Technology (1998).
- H. Aguirre, H. Okazaki, and Y. Fuwa. An Evolutionary Multiobjective Approach to Design Highly Non-linear Boolean Functions. Genetic and Evolutionary Computation Conference (GECCO) 2007, pp. 749-756.
- W. Millan, A. Clark, and E. Dawson. Heuristic design of cryptographically strong balanced Boolean functions. Advances in Cryptology - EUROCRYPT '98, 1998, pp. 489–499.
- W. Millan, A. Clark, and E. Dawson. Boolean Function Design Using Hill Climbing Methods. Information Security and Privacy, Vol. 1587 of LNCS, Springer Berlin Heidelberg, 1999, pp. 1–11.
- J. Clark and J. Jacob. Two-Stage Optimisation in the Design of Boolean Functions. Information Security and Privacy, Vol. 1841 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2000, pp. 242–254.
- J. A. Clark, J. L. Jacob, S. Stepney, S. Maitra, and W. Millan. Evolving Boolean Functions Satisfying Multiple Criteria. Progress in Cryptology - INDOCRYPT 2002, pp. 246–259.
- J. A. Clark, J. Jacob, S. Maitra, and P. Stanica. Almost Boolean functions: the design of Boolean functions by spectral inversion. CEC '03.,Vol. 3, 2003, pp. 2173–2180.

References

- L. Burnett, W. Millan, E. Dawson, and A. Clark. Simpler methods for generating better Boolean functions with good cryptographic properties, Australasian Journal of Combinatorics 29 (2004) pp. 231–247.
- J. McLaughlin and J. A. Clark. Evolving balanced Boolean functions with optimal resistance to algebraic and fast algebraic attacks, maximal algebraic degree, and very high nonlinearity, Cryptology ePrint Archive, Report 2013/011, http://eprint.iacr.org/.
- R. Hrbacek and V. Dvorak. Bent Function Synthesis by Means of Cartesian Genetic Programming. PPSN XIII, Vol. 8672 of LNCS, Springer International Publishing, 2014, pp. 414–423.
- S. Picek, E. Marchiori, L. Batina, and D. Jakobovic. Combining Evolutionary Computation and Algebraic Constructions to Find Cryptography-Relevant Boolean Functions. PPSN XIII, LNCS, Springer International Publishing, 2014, pp. 822–831.
- S. Picek, D. Jakobovic, J. F. Miller, E. Marchiori, L. Batina. Evolutionary methods for the construction of cryptographic Boolean functions. EuroGP 2015, 2015, pp. 192–204.
- S. Picek, C. Carlet, D. Jakobovic, J. F. Miller, and L. Batina. Correlation Immunity of Boolean Functions: An Evolutionary Algorithms Perspective. Genetic and Evolutionary Computation Conference (GECCO) 2015, pp. 1095–1102.
- S. Picek, R. I. McKay, R. Santana, and T. D. Gedeon. Fighting the symmetries: The structure of cryptographic Boolean function spaces. Genetic and Evolutionary Computation Conference (GECCO) 2015, pp. 457-64.

References

- L. Mariot, and A. Leporati. Heuristic Search by Particle Swarm Optimization of Boolean Functions for Cryptographic Applications. Genetic and Evolutionary Computation Conference Companion, GECCO 2015, pp. 1425–1426.
- S. Picek, S. Guilley, C. Carlet, D. Jakobovic, and J. Miller. Evolutionary Approach for Finding Correlation Immune Boolean Functions of Order t with Minimal Hamming Weight. TPNC 2015, pp. 71-82.
- L. Mariot and A. Leporati. A Genetic Algorithm for Evolving Plateaued Cryptographic Boolean Functions. TPNC 2015, pp. 33-45, 2015.
- S. Picek, D. Jakobovic, J. F. Miller, L. Batina, and M. Cupic. Cryptographic Boolean functions: One output, many design criteria. Applied Soft Computing, 40: pp. 635 - 653, 2016.
- S. Picek and D. Jakobovic. Evolving Algebraic Constructions for Designing Bent Boolean Functions. Proceedings of the 2016 on Genetic and Evolutionary Computation Conference, pp. 781-788, 2016.
- S. Picek, R. Santana, and D. Jakobovic. Maximal nonlinearity in balanced Boolean functions with even number of inputs, revisited. 2016 IEEE Congress on Evolutionary Computation (CEC), pp. 3222-3229, 2016.
- Stjepan Picek. Applications of Soft Computing in Cryptology. International Workshop on Information Security Applications 2016, pp. 305-317, 2016.
- S. Picek, D. Sisejkovic, and D. Jakobovic. Immunological algorithms paradigm for construction of Boolean functions with good cryptographic properties. Engineering Applications of Artificial Intelligence, 2016.

References

- S. Picek, C. Carlet, S. Guilley, J.F. Miller, and D. Jakobovic. Evolutionary Algorithms for Boolean Functions in Diverse Domains of Cryptography. Evolutionary Computation, MIT press, vol. 24, num. 4, pp. 667-694, 2016.
- C. Carlet. Vectorial Boolean Functions for Cryptography. In Crama, Y. and Hammer, P. L., editors, Boolean Models and Methods in Mathematics, Computer Science, and Engineering, pp. 398–469. Cambridge University Press, New York, NY, USA, 1st edition.
- J. A. Clark, J. Jacob, and S. Stepney. Searching for cost functions. CEC2004, volume 2, pp. 1517–1524.
- J. A. Clark, J. Jacob, and S. Stepney. The design of S-boxes by simulated annealing. New Generation Computing, 23 (3): pp. 219–231.
- B. Ege, K. Papagiannopoulos, L. Batina, and S. Picek. Improving DPA resistance of S-boxes: How far can we go? ISCAS 2015, pp. 2013–2016.
- J. Fuller, W. Millan, and E. Dawson. Multi-objective optimisation of bijective s-boxes. CEC 2004, volume 2, pp. 1525–1532.
- G. Ivanov, N. Nikolov, and S. Nikova. Cryptographically Strong S-Boxes Generated by Modified Immune Algorithm. BalkanCryptSec 2015, pp. 31 - 42.
- G. Ivanov, N. Nikolov, and S. Nikova. Reversed genetic algorithms for generation of bijective s-boxes with good cryptographic properties. Cryptography and Communications, 8(2): pp. 247–276.

References

- W. Millan, L. Burnett, G. Carter, A. Clark, and E. Dawson. Evolutionary Heuristics for Finding Cryptographically Strong S-Boxes. Information and Communication Security, volume 1726 of LNCS, pp. 263–274.
- S. Picek, B. Ege, L. Batina, D. Jakobovic, L. Chmielewski, and M. Golub. On Using Genetic Algorithms for Intrinsic Side-channel Resistance: The Case of AES S-box. In Proceedings of the First Workshop on Cryptography and Security in Computing Systems, CS2 '14, pp. 13 - 18.
- S. Picek, B. Ege, K. Papagiannopoulos, L. Batina, and D. Jakobovic. Optimality and beyond: The case of 4x4 S-boxes. HOST 2014, pp. 80 - 83.
- S. Picek, B. Mazumdar, D. Mukhopadhyay, and L. Batina. Modified Transparency Order Property: Solution or Just Another Attempt. SPACE 2015, pp. 210 - 227.
- S. Picek, J. F. Miller, D. Jakobovic, and L. Batina. Cartesian Genetic Programming Approach for Generating Substitution Boxes of Different Sizes. Genetic and Evolutionary Computation Companion (GECCO) 2015, pp. 1457–1458.
- S. Picek, K. Papagiannopoulos, B. Ege, L. Batina, and D. Jakobovic. Confused by Confusion: Systematic Evaluation of DPA Resistance of Various S-boxes. INDOCRYPT 2014, pp. 374–390.
- P. Tesar. A New Method for Generating High Non-linearity S-Boxes. Radioengineering, 19(1): pp. 23–26.
- S. Picek. M. Cupic, and L. Rotim. A New Cost Function for Evolution of S-Boxes. Evolutionary Computation, MIT press, vol. 24, num. 4, pp. 695-718, 2016.

References

- N. Nedjah and L. de Macedo Mourelle. Minimal Addition Chain for Efficient Modular Exponentiation Using Genetic Algorithms. Developments in Applied Artificial Intelligence. LNCS 2358, 2002, pp. 88-98.
- N. Nedjah and L. de Macedo Mourelle. Minimal Addition-Subtraction Chains Using Genetic Algorithms. Advances in Information Systems. Volume 2457 of LNCS, 2002, pp. 303 – 313.
- N. Nedjah and L. de Macedo Mourelle. Minimal Addition-Subtraction Sequences for Efficient Preprocessing in Large Window-Based Modular Exponentiation Using Genetic Algorithms. Intelligent Data Engineering and Automated Learning. Volume 2690 of LNCS, 2003, pp. 329 – 336.
- N. Nedjah and L. de Macedo Mourelle. Finding Minimal Addition Chains Using Ant Colony. Intelligent Data Engineering and Automated Learning - IDEAL 2004, pp. 642 – 647.
- N. Nedjah and L. de Macedo Mourelle. Towards Minimal Addition Chains Using Ant Colony Optimisation. Journal of Mathematical Modelling and Algorithms 5(4), 2006, pp. 525 – 543.
- N. Cruz-Cortes, F. Rodriguez-Henriquez, R. Juarez-Morales, and C. Coello Coello. Finding Optimal Addition Chains Using a Genetic Algorithm Approach. Computational Intelligence and Security. Volume 3801 of LNCS, 2005, pp. 208 – 215.

References

- N. Cruz-Cortes, F. Rodriguez-Henriquez, and C. Coello Coello. An Artificial Immune System Heuristic for Generating Short Addition Chains. Evolutionary Computation, IEEE Transactions on 12(1), 2008, pp. 1 – 24.
- L. G. Osorio-Hernandez, E. Mezura-Montes, N.C. Cortes, and F. Rodriguez-Henriquez. A genetic algorithm with repair and local search mechanisms able to find minimal length addition chains for small exponents. CEC 2009, pp. 1422 – 1429.
- A. Leon-Javier, N. Cruz-Cortes, M. Moreno-Armendariz, and S. Orantes-Jimenez. Finding Minimal Addition Chains with a Particle Swarm Optimization Algorithm. MICAI 2009: Advances in Artificial Intelligence. Volume 5845 of LNCS, 2009, pp. 680 – 691.
- N. Nedjah and L. de Macedo Mourelle. High-performance SoC-based Implementation of Modular Exponentiation Using Evolutionary Addition Chains for Efficient Cryptography. Applied Soft Computing 11 (7), 2011, pp. 4302 – 4311.
- S. Dominguez-Isidro, E. Mezura-Montes, and L.G. Osorio-Hernandez. Addition chain length minimization with evolutionary programming. Genetic and Evolutionary Computation Conference Companion, GECCO 2011, pp. 59 – 60.
- S. Dominguez-Isidro, E. Mezura-Montes, and L.G. Osorio-Hernandez. Evolutionary programming for the length minimization of addition chains. Eng. Appl. of AI 37, 2015, 125 -134.
- S. Picek, C. A. Coello Coello, D. Jakobovic, and N. Mentens. Evolutionary Algorithms for Finding Short Addition Chains: Going the Distance. EvoCOP 2016, pp. 121 – 137.

References

- C. Lamenca-Martinez, J.C. Hernandez-Castro, J.M. Estevez-Tapiador, and A. Ribagorda. Lamar: A new pseudorandom number generator evolved by means of genetic programming. PPSN IX, 2006, pp. 850-859.
- P. Peris-Lopez, J.C. Hernandez-Castro, J.M. Estevez-Tapiador, and A.Ribagorda. LAMED - A PRNG for EPC Class-1 Generation-2 RFID Specification. Comput. Stand. Interfaces 31(1), 2009, pp. 88 – 97.
- J.R. Koza. Evolving a computer program to generate random numbers using the genetic programming paradigm (1991).
- J. Hernandez, A. Seznec, and P. Isasi. On the design of state-of-the-art pseudorandom number generators by means of genetic programming. CEC2004, volume 2. pp. 1510 – 1516.
- A. Poorghanad, A. Sadr, and A. Kashanipour. Generating high quality pseudo random number using evolutionary methods. In Computational Intelligence and Security, 2008. CIS '08, pp. 331 – 335.
- L. Sekanina. Virtual reconfigurable circuits for real-world applications of evolvable hardware. Evolvable Systems: From Biology to Hardware. Springer Berlin Heidelberg, 2003, pp. 186- 197.
- S. Wolfram. Random sequence generation by cellular automata. Advances in Applied Mathematics, 7(2): pp. 123 - 169, 1986.

References

- S. Picek, D. Sisejkovic, V. Rozic, B. Yang, D. Jakobovic, and N. Mentens. Evolving Cryptographic Pseudorandom Number Generators. International Conference on Parallel Problem Solving from Nature 2016, pp. 613-622.
- S. Picek, B. Yang, V. Rozic, J. Vliegen, J. Winderickx, T. De Cnudde, and N. Mentens. PRNGs for Masking Applications and Their Mapping to Evolvable Hardware. International Conference on Smart Card Research and Advanced Applications 2016, pp. 209-227.
- S. Mangard, E. Oswald, and T. Popp. Power Analysis Attacks: Revealing the Secrets of Smart Cards (Advances in Information Security). Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2007.
- P. C. Kocher, J. Jae, and B. Jun. Differential power analysis. CRYPTO '99, 1999, pp. 388 - 397.
- R. B. Carpi, S. Picek, L. Batina, F. Menarini, D. Jakobovic, and M. Golub. Glitch it if you can: Parameter search strategies for successful fault injection, CARDIS 2013, pp. 236 -252.
- S. Picek, L. Batina, D. Jakobovic, and R. B. Carpi. Evolving genetic algorithms for fault injection attacks, MIPRO 2014, pp. 1106 – 1111.
- S. Picek, L. Batina, P. Buzing, and D. Jakobovic. Fault Injection with a new flavor: Memetic Algorithms make a difference. COSADE 2015, pp. 159 – 173.

References

- R. Maes. Physically unclonable functions: Constructions, properties and applications. Dissertation, University of KU Leuven, 2012.
- R. Maes and I. Verbauwhede. Physically unclonable functions: A study on the state of the art and future research directions. Towards Hardware-Intrinsic Security, 2010, pp. 3-37.
- G. T. Becker. The gap between promise and reality: on the insecurity of XOR arbiter PUFs. International Workshop on Cryptographic Hardware and Embedded Systems – CHES 2015, pp. 535-555.
- A. Heuser, S. Picek, S. Guilley, and N. Mentens. Side-channel Analysis of Lightweight Ciphers: Does Lightweight Equal Easy? Cryptology ePrint Archive, Report 2017/261, 2017.
- A, Heuser and M. Zohner. Intelligent machine homicide. International Workshop on Constructive Side-Channel Analysis and Secure Design, pp. 249-264, 2012.
- L. Lerman, G. Bontempi, and O. Markowitch. Side channel attack: an approach based on machine learning. Center for Advanced Security Research Darmstadt, pp. 29-41, 2011.
- L. Lerman, G. Bontempi, and O. Markowitch. A machine learning approach against a masked AES. Journal of Cryptographic Engineering, vol. 5, num. 2, pp. 123-139, 2015.