Evolutionary algorithms with and without adaptive mutation in AI based cryptography

Mateusz Tybura[1,*]

[1] Rzeszow University of Technology, Faculty of Electrical and Computer Engineering, Rzeszow, Poland

Abstract. The key role of cryptography is to make cipher so hard to reproduce without knowing all the details that no one besides the recipient could decipher the message. Those algorithms which are used nowadays gets its security mostly from highly reliable algorithms and/or complicated cryptographic keys. Unfortunately, those human-made methods aren't invulnerable so sooner or later they compromise. So, it could be really useful to make a cipher which could change. But currently only neural networks are capable of thing known as transfer learning. In this article similar method was proposed in order to make it possible to re-learn already established evolutionary algorithm to do new, similar task.

1 Introduction

Cryptography (gr. *kryptos* – hidden, *graphein* – write) a field of knowledge dedicated to hiding data has no doubly concerned peoples through millennials. Mostly because it possibly started as soon as some of primitive cultures had developed written languages. Over the years it evolved from very primitive textual ciphers to those used on computers. Modern solutions could be easily divided into three categories: highly mathematical, device based, intelligent based.

Those highly mathematical are using more less sophisticated theories like very large prime numbers, elliptic curves, chaotic equations or many others. For instance, RSA and ECC are both using very large prime numbers, with the security highly dependent on problem known as discrete logarithm [1]. But because of need for extensive calculations they are used mostly for establishing connection between peers to then exchange keys for less complex, symmetric ciphers [2].

Another way of doing cryptography is to use dedicated devices instead of running any kind of written program on CPU [3-5]. That kind of cipher chips are built with either programmable circuits such as FPGA [4,5] or specially designed security processors. Such electronic circuits are currently widely used in smartphones. It for sure makes it faster and less vulnerable for some certain types of attacks but more prune to the others. Cause as usual even very promising solutions have their flaws. Electronic devices had to be designed not only for doing was is really needed to do but also to not expose a lot of details of their internal processes. Many times, it was overlooked. For instance, some of devices had given

* Corresponding author: tyburam@hotmail.com

a great opportunity to get some internal details of what they are doing just by registering the heat map [5]. Additionally, it's sometimes impossible to change the way the electronics work if it was not designed for making it possible.

The most modern and promising development direction is to use intelligent based techniques to make cryptography algorithms [6-15]. For many years both neural networks and evolution algorithms were used to make ciphers as well as to break ciphers. For example, in 1995 neural networks were used to recreate DES algorithm. That kind of researches are also done in big IT companies such as Google. In spite of that fact this article will be strictly focusing on the usage of intelligent algorithms in the cryptography.

2 Evolutionary algorithms in cryptography

Application of evolutionary algorithms in cryptography relies on the idea of synchronized learning on two separate machines. Which insist on having the same learning configuration, training data and randomness. Making more or less mistakes in setting the same training environment could make it impossible for usage in any kind of real problem. Using those techniques to generate ciphers seems to be good idea as those generated ones aren't as known as the already known ones. Unfortunately, it could also lead to some hidden defects for the same reason. So, they must be treated as a double-sided blade.

An interesting fact is that in scientific articles and theses there are way more usage of neural networks instead of evolutionary algorithms in cryptography itself. Which could be the outcome of overall higher popularity of for instance deep neural networks or GANs and things like that. Also, it seems that the evolutionary algorithms had been used more frequently in both analysing and breaking ciphers instead of making ones. Many researches had shown that they could be used to decipher a message encoded using algorithms such as Vigenere [7] or DES [9]. For the generation task there are methods such as CIGA (Cryptography Inspired by Genetic Algorithms) or ICIGA (Improved CIGA) [14].

3 Instruction set based evolutionary ciphers

Plain genetic algorithm was chosen for the first experiments. Implemented in a way shown on listing 1. It served as a baseline for further comparison and improvement.

Listing 1. Psuedocode for evolution.

```
function evolution
    pop = generate_initial_population(size)
    foreach individual in pop do
        individual.fit = fitness(individual)
    while not stop(pop) do
        offsprings = select(pop, sel_size)
        offsprings = crossover(offsprings)
        offsprings = mutate(offsprings)
        if size(offspringa) < size then
            offspring = fill(offspring, size)
        pop = offspringa
    return best(pop)
```

Every single individual was represented by an integer number encoded instruction. Those instructions were four arithmetic ones (addition, substitution, multiplication,

division). All of them operating only on integer numbers. Also, their deterministic nature as well as elementary nature in sense of being implemented in every single modern CPU guarantied multiplatform usability.

First task taken into consideration was to achieve the best possible accuracy in pseudo Caesar cipher, Caesar cipher and some made up substitution cipher with hardcoded shift.

The evolutionary way was much more successful in both correctness and time for getting answer. For instance, when generating pseudo Caesar cipher the right answer arrived just after generating initial population, when there were 20 individuals, just 4 possible operations (add, sub, mul and div) and fitness function which simple checked for the effect of using generated cipher. Adding more operations made the task harder but not as much as when the operations didn't have the addition inside. But there is an exception for the time when the initial population doesn't have even a single individual with a right gene and the mutation probability is set something smaller than 0,5. In that case there's no way of getting even approximated answer, as there's no way to get the right set of genes even after thousands of generations. Thus, it is absolutely necessary to choose genes representing operations which are performed by the algorithm being mimicked and the fitness function really checking for what is needed to be checked and the right way to interpret the genes of the best individual. If not nothing good have happened.

4 Adaptive mutation

As it was already shown in many researches' method named transfer learning could make it possible to reuse pretrained network for more less similar task. But there were many questions. Is it possible to use it when working with ciphers? Do they have to work really similar? And how to make something like transfer learning using evolutionary algorithms instead of neural network?

In the experiments three distinct methods were proposed. First was to just continue learning with no changes in algorithm at all. Unfortunately, every single time the evolutionary algorithms failed. Secondly it was adjusted just a little bit. The evolution was stopped before getting better than about 70% or 50% or any other arbitrary chosen value of fitness function achieved while learning the first cipher but it couldn't really make it. Hence it was clear that just stopping isn't enough to make a genetic adaptation of transfer learning.

It was assumed that changes in parameter such as mutation rate could possibly be the best way into getting better results as that's the only way to change genes in population to those that aren't currently represented in it. So, after the initial failure, new possibilities were tested in order to check which will do the best job for this certain task.

For the first time the algorithm was almost the same but first it had to stop right before getting into 70% of accuracy and two the mutation rate was increased to value 0,5 for the rest of the evolution. It didn't work well. When the best individuals were analysed it something interesting had been found. Just changing mutation rate didn't worked because of introducing to more randomness instead of helping (fig. 1). Hence it seemed to be reasonable to make more intelligent mutation rate changing routine.

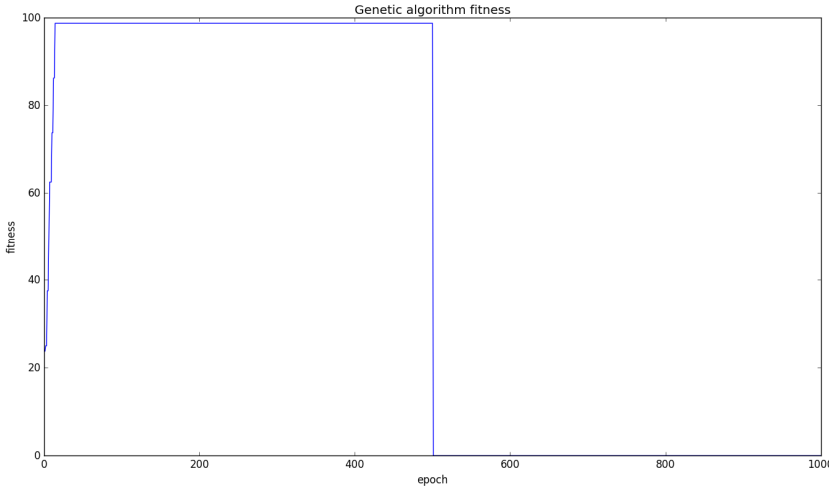

Fig. 1. Evolutionary algorithm fitness function value after switching task and changing mutation rate

In the last attempt special mutation kind was proposed. After a thoughtful process it became clear that this mutation must change a lot of genes in the less fitted ones and at the same time limit the changes made to the best ones. With something like this it These trials had shown that it could work but only when performed well. The most promising one consisted of few main things combined into adaptive mutation method (listing 2).

Listing 2. Pseudocode for adaptive mutation

```
function adaptive_mutate(individuals)
    max_fit = best(individuals).fit
    if max_fit == 0 then
        max_fit = 1
    mutated = <empty_list>
    foreach ind in individuals do
        prob = (max_fit - ind.fit) / max_fit
            if random() >= prob then
                mutated.add(mutate(ind))
     return mutated
```

Proposed method started with checking for the maximum value of fitness function in all individuals available for mutation. If for some reason it is equal to zero, it is adjusted to 1 to avoid errors when dividing by zero. Then for every single individual mutation probability is set to the difference between maximal fitness and fitness of current individual divided by maximal fit. So, the most fitted ones aren't mutated and the less fitted are most likely to be changed. Hence it at the same time tries to avoid losses by not mutating best ones and increases diversity by mutating the worst ones. For experimental set it made way better than plain evolutionary algorithm or that with randomly changed mutation probability (fig. 2).

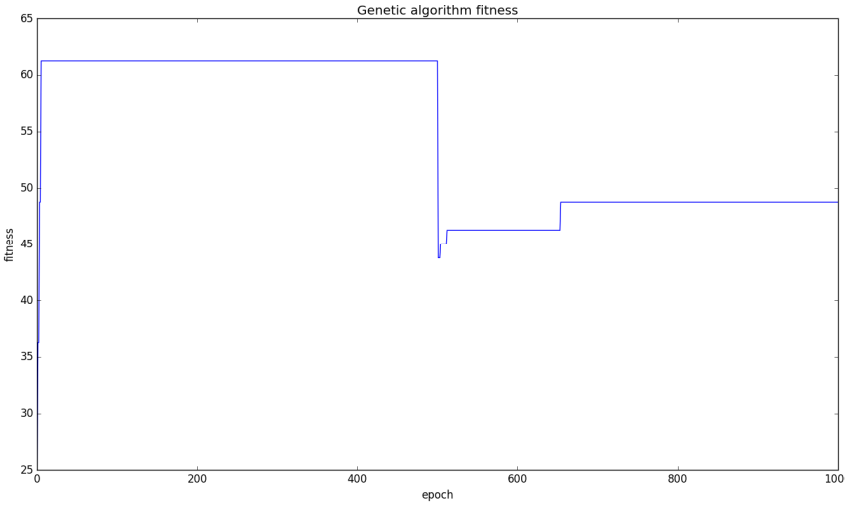

Fig. 2. Evolutionary algorithm fitness function value after switching task and changing mutation rate in adaptive manner

This method had made a huge difference (fig. 2). While still it hadn't made the best possible score, it had changed a lot. Because of removing the randomness and incorporating the adaptiveness the whole population doesn't change so drastically resulting in 0 as a value of the fitness function. Then the fitness even managed to change a little bit in a good way.

Summary

Evolutionary algorithm could be used for making cipher without much of human interventions as in fully human-made solutions. They ability to learn makes quality outcome in just a matter of time. While standard approach makes it difficult to change mind, the adaptive one plays a huge role in making more sophisticated, more adaptive solution.

References

1. Smart N.: Cryptography: An Introduction, McGraw-Hill College, 2004.
2. TLS protocol specification, https://tools.ietf.org/html/rfc5246
3. Ay A. U., Ozturk E., Henriquez F. R., Savas E.: Design and implementation of a constant-time FPGA accelerator for fast elliptic curve cryptography, International Conference on ReCOnFigurable Computing and FPGAs, 2016
4. Bartlik M., Bucek J.: A low-cost multi-purpose experimental FPGA board for cryptography applications, 4th Workshop on Advances in Information, Electronic and Electrical Engineering, 2016
5. Gurkaynak F. K.: GALS System Design: Side Channel Attack Secure Cryptographic Accelerators. PhD thesis, ETH Zurich, 2006

6. Mahmood A., Dony R., Areibi S.: An adaptive encryption based genetic algorithms for medical images, IEEE International Workshop on Machine Learning for Signal Processing, 2013
7. Omran S. S., Al.-Khalid A.S., Al-Saady D. M.: A cryptoanalytic attack on Vigenere cipher using genetic algorithm, IEEE Conference on Open Systems, 2011
8. Feng-Tse L., Cheng-Yan K.: A genetic algorithm for ciphertext-only attack in cryptanalysis, IEEE International Conference on Systems, Man and Cybernetics, 1995
9. Jun S., Huanguo Z., Qingshu M., Zhangyi W.: Cryptanalysis of Four-Round DES Based on Genetic Algorithm, International Conference on Wireless Communications, Networking and Mobile Computing, 2007
10. Zarza L., Pegueroles J., Soriano M.: Evaluation Function for Synthesizing Security Protocols by means of Genetic Algorithms, The Second International Conference on Availability, Reliability and Security, 2007
11. Albassall A. M. B., Wahdan A.-M.A.: Genetic algorithm cryptanalysis of a feistel type block cipher, International Conference on Electrical, Electronic and Computer Engineering, 2004
12. Ribaric T., Houghten S.: Genetic programming for improved cryptanalysis of elliptic curve cryptosystems, IEEE Congress on Evolutionary Computation, 2017
13. Ho Yean L., Samsudin A., Belaton B.: Heuristic cryptanalysis of classical and modern ciphers, 13th International Conference on Networks jointly held with 7th Malaysia International Conference on Communcation, 2005
14. Picek S., Golub M.: On evolutionary computation methods in cryptography, Proceedings of 34th International Convention MIPRO, 2011
15. Clark A. J.: Optimisation Heuristics for Cryptology, PhD thesis, Information Security Research Centre, Faculty of Information Technology, Queensland University of Technology, 1998