UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING
(FER)

BACHELORS THESIS no. 3488

SOLVING CRYPTOGRAPHIC PRIMITIVES OPTIMIZATION PROBLEM USING EVOLUTIONARY COMPUTATION ALGORITHMS

Nicole Bilić

Zagreb, June of 2014.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA ZAVRŠNI RAD MODULA

Zagreb, 10. ožujka 2014.

ZAVRŠNI ZADATAK br. 3488

Pristupnik: **Nicola Bilić**
Studij: Računarstvo
Modul: Računarska znanost

Zadatak: **Solving cryptographic primitives optimization problem using evolutionary computation algorithms**

Opis zadatka:

Examine methods for evaluating cryptographic primitives, especially regarding S-boxes. Enumerate and describe properties of S-boxes. Study evolutionary computing algorithms convenient for solving S-boxes optimization problems. Using ECF (Evolutionary Computation Framework) programming environment, implement program system for optimization of S-boxes using arbitrary evolutionary algorithms. Test system functioning on S-boxes for AES cryptosystem. Examine application of program system for finding smaller size S-boxes for cryptosystems suited for embedded systems.

Zadatak uručen pristupniku: 14. ožujka 2014.
Rok za predaju rada: 13. lipnja 2014.

Mentor:

Izv.prof.dr.sc. Marin Golub

Djelovođa:

Doc.dr.sc. Tomislav Hrkać

Predsjednik odbora za završni rad modula:

Prof.dr.sc. Siniša Srbljić

Content

Introduction

Cryptology is a scientific discipline concerned with the study of codes and ciphers and techniques for the protection of communication in presence of a third party (adversaries) [2]. It is a science concerned with data communication and storage in secure and usually secret form. It engages with protocol analysis and construction, in order to overcome the influence of adversaries. Moreover, cryptology is, in modern society, strongly involved with information security which is one of the most important problems awaiting a solution.

There are two main fields in cryptology:

1. Cryptography - invention of encryption algorithms such that it makes it impossible for any adversary to break them. Modern cryptography has four objectives:

    a) Confidentiality – the information can be understood only to who it was intended to.

    b) Integrity – the information comes to its destination without altering in storage or transition between sender and intended receiver. However, if it changes in any matter, the change has to be detected.

    c) Non-repudiation – the sender of the information cannot later deny his intention of creation or transmission of the information.

    d) Authentication – both sender and receiver can confirm each other's identity and origin/destination of the information.

2. Cryptanalysis – study of ciphers, ciphertext or cryptosystems with a view to finding weaknesses in them that will permit retrieval of the plaintext from the ciphertext, without necessarily knowing the key or the algorithm. There is a great number of ways to perform cryptanalysis, such as: Known-plaintext analysis, Ciphertext-only analysis, Man-in-the-middle attack,

Timing/differential power analysis. For more information refer to [5].

In this paper we are going to discuss S-boxes and their properties, as well as evolutionary computing algorithms convenient for solving S-boxes optimization problems using ECF (Evolutionary Computation Framework). The focus will be on S-boxes for AES cryptosystem.

The core part of cryptographic operation is a cryptographic key. It is a variable value that is applied using an algorithm to a string or block of unencrypted text, or to decrypt encrypted text [5]. Encryption algorithms are divided into two big groups - symmetric key algorithms which use the same key for both encryption and decryption and asymmetric key algorithms whose encryption and decryption keys differ. In this paper, we are more interested in symmetric key algorithms, being that AES belongs in this category. More precisely, it belongs to its subclass – block ciphers.

Block cipher algorithms are vulnerable to various kinds of cryptanalysis, such as, more traditional linear and differential attacks as well as more modern side-channel attacks. The aim of the S-box, in the case of AES cryptosystem, is to hide the correlation between input and output bits. Small cryptographic devices such as smart cards, RFID tags, etc. are commonly used today and they store big amounts of our personal data that must be secure. Algorithms for their security are stored either in software or hardware on those physical devices. These devices cause unintentional output channels which are called side channels. The best known and most commonly used side channel is power consumption. If the adversary is able to collect many power consumption traces, she can use powerful statistical and mathematical methods to recover the key from the time series data. This type of attack is called Differential Power Analysis - DPA. Side channel attacks are the main security threat for smart cards [1].

1. Boolean functions and properties

Boolean functions can, in the context of cryptography, be represented by truth tables, algebraic normal form, numerical normal form and trace representation [3]. We will discuss the properties and definitions of vectorial Boolean functions (S-boxes).

1.1. Boolean functions

A Boolean function *f* is a mapping from F_2^n into F_2. Boolean functions are a special form of S-boxes with only one output variable. In mathematics it is usually defined as a mapping from $\{0,1\}^n$ to {0,1}. Therefore, in order to learn about S-boxes, we have to study some of the properties of Boolean functions.

The Hamming weight HW(*f*) of a Boolean function *f* is the number of ones in its binary truth table.

A Boolean function *f* on F_2^n can be represented uniquely by a truth table(TT), which is a vector (*f*(0),....,*f*(1)) that contains the function values of *f*, ordered lexicographically.

Structure of the TT can be:

1. Normal – a Boolean function *f* is said to be k-normal for k ∈ {1,...,n} if there exists a flat *V* of dimension k such that *f* is constant on V. If $k = \left\lceil \frac{n}{2} \right\rceil$ the function is said to be normal.

2. Weakly normal – The same definition as normality, but the function is affine instead of constant when restricted to the flat.

Associated with the Boolean function *f* is the function $(-1)^f = 1 - 2f$ whose function values belong to the set {1, -1}[1] [3].

The most important mathematical tool for the study of cryptographic properties of Boolean functions is the Walsh (or Hadamard) transform.

The Walsh transform permits to measure the correlation between a Boolean function and all linear Boolean functions. The knowledge of the Walsh transform of a Boolean function uniquely determines the function and hence it is possible to work entirely with the Walsh transform. Its systematic use leads to uniform, elegant and efficient treatments and statements of the main cryptographic criteria. Resiliency and nonexistence of linear structures are directly related to the properties of the support of the Walsh transform of a Boolean function, i.e. its Walsh support [7].

The Walsh transform gives information on the linearity of the functions, while the autocorrelation function is related with the differential properties of the function.

1.1.1. Walsh transform

In mathematical means, Walsh transform is an example of a generalized class of Fourier transforms. It performs orthogonal, symmetric, involutional, linear operation on 2^m real numbers. The Walsh transform can be regarded as being built out of size-2 discrete Fourier transforms (DFTs) and is in fact equivalent to a multidimensional DFT of size 2 x 2 x … x 2 x 2. It decomposes an arbitrary input vector into a superposition of Walsh functions [2].

Regarding Boolean functions and Walsh transform, Boolean function *f* on F_2^n is uniquely determined by its Walsh transform. The Walsh transform, denoted by W_f, of *f* is a real-valued function defined for all $\bar{\omega} \in F_2^n$ as [3]:

$$W_f(\bar{\omega}) = \sum_{\bar{x} \in F_2^n} (-1)^{f(\bar{x}) \oplus \bar{x} \cdot \bar{\omega}} = 2^n - 2wt(f \oplus \bar{x} \cdot \bar{\omega}) \qquad [1.1]$$

1.1.2. Autocorrelation Function

Autocorrelation, also known as serial correlation, is the cross-correlation of a signal with itself. It is the similarity between observations as a function of the time lag between them. It is a mathematical tool for finding repeating patterns, such as the presence of a periodic signal obscured by noise, or identifying the missing fundamental frequency in a signal implied by its harmonic frequencies [2].

In the matter of cryptography and Boolean functions, the autocorrelation function of the Boolean function f on F_2^n is a real-valued function defined for all $\bar{\omega} \in F_2^n$ [3]:

$$r_f(\bar{\omega}) = \sum_{\bar{x} \in F_2^n} (-1)^{f(x) \oplus f(x \oplus \bar{\omega})} \qquad [1.2]$$

but does not uniquely determine the function in contrast to Walsh transform [3].

1.1.3. Properties related with Walsh Spectrum

Only the properties related to the practical part of this thesis are going to be described.

For a Boolean function to be acceptable for the use in cryptography, it has to be balanced, with high nonlinearity, high algebraic degree, high correlation immunity and low global avalanche characteristics. In order to accomplish better resistance against side-channel analysis, the transparency order should be as low as possible.

Since it has been proved that minimal transparency equals 0, which is only the case for linear and affine functions, two problems are introduced. Linear and affine functions have nonlinearity equal to zero and therefore are not acceptable for cryptographic usage.

First problem is to find an appropriate level of nonlinearity and the second one is to find a Boolean function with at least that level of nonlinearity and optimal transparency order for that specific case [4].

1. Balancedness – A Boolean function is balanced if its output is equally distributed, i.e., its weight is equal to 2^{n-1}, i.e. the number of ones equals the number of zeros in the truth table. This translates in Wf(0) = 0 for the Walsh spectrum [3].

2. Nonlinearity – The nonlinearity of a Boolean function *f*, denoted by N_f is defined as the minimum Hamming distance to the nearest affine function on F_2^n, i.e.:

$$N_f = min_{g \in RM(1,n)} d(f,g) \qquad [1.3]$$

The nonlinearity of *f* can be expressed in terms of the Walsh coefficients by [3]:

$$N_f = 2^{n-1} - \frac{1}{2} max_{\bar{\omega} \in F_2^n} |W_f(\bar{\omega})| \qquad [1.4]$$

3. Transparency order – cryptographic property of S-boxes introduced by Prouff in 2005. For (*n*,*m*)-function it can be defined as follows [3]:

$$T_f = max_{\vec{\beta} \in F_2^n} \left(\left| m - 2HW(\vec{\beta}) \right| - \frac{1}{2^{2n} - 2^n} \sum_{a \in F_2^{n*}} \left| \sum_{\substack{\vec{v} \in F_2^n \\ HW(\vec{v})=1}} (-1)^{\vec{v} \cdot \vec{\beta}} W_{D_a f}(\vec{0}, \vec{v}) \right| \right) \qquad [1.5]$$

where W_{Daf} represents Walsh transform of the derivative of *f* with respect to a vector $a \in F_2^n$.

Other properties related with Walsh Spectrum are: plateaued functions, correlation immunity, resiliency, bias of nonlinearity, linearity and homomorphicity. More information on these properties can be found in [3].

1.1.3.1 Example 1.

S-box s = {4 1 3 5 7 0 6 2}
Input size M = 3
Output size N = 3
S-box is balanced.
Nonlinearity = 4.
Transparency order = 2.571.

1.1.4. Properties related with Autocorrelation Spectrum

Derivative – The function $D_{\bar{\omega}}f(\bar{x}) = f(\bar{x}) \oplus f(\bar{x} \oplus \bar{\omega})$ is called the derivative of f with respect to the vector $\bar{\omega}$.

Propagation characteristics – A function is said to satisfy the propagation characteristics of degree *l*, denoted by PC(*l*), if all its derivatives w.r.t.vectors $\bar{\omega}$ with $1 \leq wt(\bar{\omega}) \leq l$ are balanced.

SAC (strict avalanche criterion) – The property PC(1) is also called strict avalanche criterion.

Linear structure – If the derivative is a constant function, the vector $\bar{\omega}$ is called a linear structure of *f*.

GAC (global avalanche characteristics) – The global avalanche characteristics indicators consist of an absolute indicator, defined by

$$D_\infty(f) = max_{\omega \in F_2^n \setminus \{0\}} \left| r_f(\bar{\omega}) \right|$$

and a sum-of-square indicator defined by

$$D_2(f) = \sum_{\omega \in F_2^n} r_f(\bar{\omega})^2$$

For more information on these properties look [3].

1.1.5. Bent functions

In the mathematical field of combinatorics, a bent function is a special type of Boolean function. It takes several inputs and gives one output, each of which has two possible values (such as 0 and 1, or true and false). Bent functions are called so because they are as different as possible from all linear and affine functions. This makes the bent functions hard to approximate [2]. This is one of the main reasons why bent functions are studied in cryptography.

Bent functions, as one of the important terms in context of cryptography and Boolean functions, must satisfy following properties [3]:

1. The Walsh spectrum is flat, i.e., $\left| W_f(\bar{\omega}) \right| = 2^{n/2}$ for all $\bar{\omega} \in F_2^n$.
2. The function *f* has maximum nonlinearity equal to $2^{n-1} - 2^{(n/2-1)}$

3. The 2^n x 2^n matrices $N = (n_{i,j})$ and $M = (m_{i,j})$ with

$$n_{i,j} = \frac{1}{2^n} W_f(\bar{x}_i \oplus \bar{x}_j) \qquad [1.6]$$

and

$$m_{i,j} = (-1)^{f(\bar{x}_i \oplus \bar{x}_j)} \qquad [1.7]$$

for $0 \leq i,j \leq 2^n - 1$ are Hadamard matrices[1].

4. The function satisfies PC(n), i.e. $r_f(\bar{\omega}) = 0$ for all $\bar{\omega} \neq 0$

5. The support of *f* is a difference set [3].

In other words, bent functions are functions that have maximal nonlinearity, but they are not appropriate in cryptography since they are not balanced.

1.2. Vectorial Boolean Functions or S-Boxes

S-box, or a substitution box, is an (*m* x *n*) function, where *m* is a number of an input bits and *n* is a number of an output bits. Furthermore, *m* can, but doesn't have to be equal to *n*. Substitution box is used as a transformation function between input and output bits. It finds its main purpose in symmetric key algorithms where it's used to hide the relation between the key and an output ciphertext. S-boxes can be implemented in many ways but, within ECF and this paper are mostly implemented as look-up tables, either fixed or dynamically generated from the key.

In the theory of block ciphers and linear cryptanalysis, linear approximation table is studied with respect to the operation $\oplus$. The linear approximation table is a 2^n x 2^m table whose entries are defined as [3]:

$$L_{a,b} = \#\{\bar{x}: \bar{b} \cdot F(\bar{x}) = \bar{a} \cdot \bar{x}\}, \forall 0 \leq b \leq 2^m, \forall 0 \leq a < 2^n \qquad [1.8]$$

---

1. Hadamard matrix – square matrix whose entries are either +1 or -1 and whose rows are mutually orthogonal[2]

The differential table is a 2^n x 2^m table whose entries are defined as [3]:

$$D_{a,b} = \#\{x : F(x) \oplus F(x \oplus a) = b\}, \forall 0 \leq b \leq 2^m, \forall 0 \leq a \leq 2^n \qquad [1.9]$$

The nonlinearity and linearity of an S-box *F* are defined as [3]:

$$N_F = min_{\bar{b} \in F_2^n \setminus \{0\}} N_{\bar{b} \cdot F}, L_F = min_{b \in F_2^n \setminus \{0\}} L_{\bar{b} \cdot F}. \qquad [1.10]$$

The Walsh (or autocorrelation) spectrum of *F* is defined as the collection of all Walsh (autocorrelation) spectra of the component functions of *F*.

The relation between the entries of the differential table and the autocorrelation coefficients of the linear combinations of the components

b · *F* is made by a Fourier transform [3]:

$$r_{\bar{b} \cdot F}(\bar{a}) = \sum_{x \in F_2^n} (-1)^{\bar{b} \cdot \bar{x}} D_{a,x} \text{ and } D_{a,b} = 2^{-n} \sum_{\bar{y} \in F_2^m} (-1)^{\bar{b} \cdot \bar{y}} r_{\bar{y} \cdot F}(\bar{a}) \qquad [1.11]$$

Another important parameter related to S-box is the algebraic degree of the (*n,m*) S-box F which is defined as [3]:

$$\deg(F) = max_{b \in F_2^n} \deg(\bar{b} \cdot F) = max_{i \in \{0,\ldots,n-1\}} \deg(\bar{e}_i \cdot F) \qquad [1.12]$$

In the case of AES (Advanced Encryption Standard), special subclass of S-box is used – Rijndael S-box. Rijndael S-box has the same number of input and output bits (*n* equals *m*) so it is represented as a square matrix. In the design of such S-box, it is attempted to accomplish resistance to linear and differential cryptanalysis. That is accomplished by manipulating S-box properties, some of which are in inversely proportionate relation, in order to find an optimal solution. AES S-box size is 8x8 (8 bits of input, 8 bits of output).

A vectorial Boolean function $F_2^n \rightarrow F_2^m$ (or S-box) can be represented by the vector $(f_0, f_1 \ldots . f_{m-1})$ where f_i are Boolean functions from F_2^n into F_2 for $0 \leq i \leq m-1$, which we also call the component or output functions of the S-box [3].

The aim is to find an optimal solution, i.e. the best possible rate between transparency order and nonlinearity. In order to find an optimal

S-box for AES encryption, within this paper, we take into consideration three properties: balancedness, nonlinearity and transparency order. Using ECF and applying different genetic algorithms we try to evolve the best solution possible taking into account these three properties.

2. Tools and algorithms

In the practical part of this paper, comparison of achieved results, using different genetic algorithms, is performed. These specific algorithms are part of an Evolutionary Computation Framework, and optimization of

S-boxes is only one of the potential uses of this framework. Algorithm, along with its parameters is read-in from a configuration XML file. This file is used to operate with parameters of particular algorithms as well as to set other parameters important for ECF and S-box. We are interested in balanced

S-boxes with as low transparency order as possible and with large, but not necessary optimal nonlinearity.

```
<ECF>
        <Algorithm>
                <SteadyStateTournament>
                        <Entry key="tsize">3</Entry>
                </SteadyStateTournament>
                <RouletteWheel>
                        <Entry key="crxprob">0.5</Entry>
                        <Entry key="selpressure">10</Entry>
                </RouletteWheel>
        </Algorithm>

        <Genotype>
                <Permutation>
                        <Entry key="size">256</Entry>

                        <Entry key="crx.COSA">1</Entry>
                        <Entry key="crx.cyclic">1</Entry>
                        <Entry key="crx.cyclic2">1</Entry>
                        <Entry key="crx.DPX">1</Entry>
                        <Entry key="crx.OPX">1</Entry>
                        <Entry key="crx.OX">1</Entry>
                        <Entry key="crx.OX2">1</Entry>
                        <Entry key="crx.PBX">1</Entry>
                        <Entry key="crx.PMX">1</Entry>
                        <Entry key="crx.SPX">1</Entry>
                        <Entry key="crx.ULX">1</Entry>
                        <Entry key="crx.UPMX">1</Entry>

                </Permutation>
        </Genotype>

        <Registry>
                <Entry key="population.size">100</Entry>
                <Entry key="term.stagnation">50</Entry>
                <Entry key="term.maxgen">200</Entry>
                <Entry key="log.level">3</Entry>
                <Entry key="log.filename">log.txt</Entry>
                <Entry key="batch.repeats">1</Entry>
                <Entry key="batch.statsfile">stats.txt</Entry>
        </Registry>
```

Figure 2.1 Configuration file for ECF

The program uses the specified algorithm, and in the case of multiple algorithms found, takes the first one. For each algorithm, specific options can be set. For Steady State Tournament the only possible option to set is *tournament size*. On the other hand, Roulette Wheel has two options – *crxprob* and *selpressure* which are to be explained in the following part of the paper.

It is possible also to select different genotypes. For the purpose of

S-boxes, we use *Permutation* as genotype. Within the *Registry* tags it is possible to set options that are common for all the algorithms – population size, maximum number of generations etc.

2.1. Evaluated properties of S-boxes

As mentioned above, properties of S-box that are of interest are balancedness and nonlinearity. Other then that, there is also transparency order. These three properties create the fitness value of each individual

(S-box).

1. Balancedness

Equal number of zeros and ones for each of Boolean functions and for linear combinations of them.

2. Nonlinearity

Smallest Hamming distance between Boolean function and its best affine approximation and for linear combinations of all Boolean functions.

3. Transparency order

2.2. Properties calculation within ECF

2.2.1. Balancedness

```
int balance (u8 *tt) //this is maybe better with walsh transform coefficients
{
    int i, rez=0;

   for(i=0; i < (1 << M); i++)
   {
       if (tt[i] == 0) //count zeros
           rez=rez+1;
   }
   if (rez == ((1 << (M-1))))
       return 0; //if the Boolean functions is balanced, we give nothing since it is constraint
   else
       return 1;
}
```

Figure 2.2 Method for balancedness calculation[ECF]

It is obligatory for every S-box to be balanced. Therefore, balancedness is checked for every individual of generation. In order to do so, we need to count zeroes and/or ones and compare it to the number of zeroes/ones on an entrance to an S-box.

2.2.2. Nonlinearity

For (*n,n*) S-boxes F, $\Delta_F \geq 2$ and F is an almost perfect nonlinear function(APN) if the equality is satisfied. For (*n,n*) S-boxes F with n odd, the inequality $L_f \geq 2^{(n+\frac{1}{2})}$ holds, and F is called almost bent if the lower bound is reached. It is proven that an almost bent function is almost perfect nonlinear, but the converse is not always true [3].

So in the case of AES S-boxes (*n* = 8), we expect to find functions with nonlinearity below 120, i.e. in case of an AES, nonlinearity is from an interval [0,120].

```
int wt_nl (u8 *tt) //nonlinearity of the Boolean function
{
    int i, j, m, halfm, t1, t2, r, a, b, max = 0;

    int *rez=0;

    rez=(int *) malloc((1 << M)*sizeof(int));

    for(i=0; i < (1 << M); ++i )
        rez[i] = (tt[i] == 0)? 1 : -1;

    for( i = 1; i <= M; ++i ) {
        m  = (1 << i);
        halfm = m/2;
        for( r=0; r < (1 << M); r += m ) {
            t1 = r;
            t2 = r + halfm;
            for( j=0; j < halfm; ++j, ++t1, ++t2 ) {
                a = rez[t1];
                b = rez[t2];
                rez[t1] = a + b;
                rez[t2] = a - b;
                //help so I don't need to search for maximum later
                if (abs(rez[t1]) > max)
                    max = abs(rez[t1]);
                if (abs(rez[t2]) > max)
                    max = abs(rez[t2]);
            }
        }
    }

    free(rez);

    return nonlinearity (max);
}
```

Figure 2.3 Method for nonlinearity calculation [ECF]

2.2.3. Transparency order

Transparency order is very important property of S-boxes related with the resistance of S-boxes to the differential power analysis attacks. The higher the transparency order is, the lower is the S-box resistance to the DPA attacks.

Since it has been established that the lowest possible transparency order is zero and it is defined for affine and linear functions which are not of interest in cryptography usage, the goal is to find S-box with lowest possible level of transparency order that is greater than zero. For AES S-boxes, transparency order is from an interval [0,8].

```
float transparency2 (u8 *tt)
{
    float res = 0.0;
    float C = (float) ((1 << (2*N)) - (1 << N));
    float K = (float) (N * (1<<N));
    float z = 0;
    int b, a, x, tmp1, tmp2;
    float sigma1 = 0, sigma2 = 0;
    int tmp = 0;

    tmp1 = (1 << M);
    tmp2 = (1 << N);


    for (b = 0; b < tmp2; b++)
    {
        sigma2 = 0;
        for (a = 1; a < tmp2; a++)
        {
            sigma1 = 0;
            for (x = 0; x < tmp2; x++)
            {
                sigma1 = sigma1 + hammingWeight(b^(evaluate_box(tt,x,a)));
            }
            z = K - 2*sigma1;
            if (z < 0)
                z = z*(-1);
            sigma2 = sigma2 + z;
        }
        tmp = 2 * hammingWeight(b);
        if (res < abs(M - tmp)-sigma2/C)
            res = abs(M - tmp)-sigma2/C;
    }

    return res;
}
```

Figure 2.4 Method for transparency order calculation[ECF]

2.3. Fitness functions

Fitness functions are special functions used to guide the evolution process in an appropriate direction. Fitness function, in case of S-boxes, takes into account three properties described earlier. It could take into account more different properties, but regarding the tradeoff between them, it is for the best that fitness function stays as simple as possible.

Taking into account that unbalanced Boolean functions are not acceptable in cryptography, we strictly insist on balancedness property to be a part of a fitness function. In one case, where the primary aim is high nonlinearity and secondary transparency order, the fitness function is defined as [4]:

$$fitness1 = BAL + NL_f + (1 - T_f)$$

The balancedness property (BAL) is used as a penalty and calculated as [4]:

if $(h_\omega(TT) > \frac{2^n}{2})$

then

$$BAL \leftarrow \frac{2^n - h_\omega(TT)}{2^n}\bar{X}$$

else

$$BAL \leftarrow \frac{2^n}{2^n - h_\omega(TT)}\bar{X}$$

end if

where it has experimentally been determined that X = -5 scales well for Boolean functions with n = 8 inputs. Balanced function receives the value 1, and unbalanced functions receive a negative value corresponding to the level of imbalance [8]. The optimization problem considers the maximization of the fitness function.

In other case, the goal is to find as low transparency order as possible for the given level of nonlinearity.

Fitness function is defined as [4]:

$$fitness2 = BAL + (1 - T_f) - pos(2 \times (NL_t - NL_f))$$

where the function pos(x) returns x if x>0 and zero otherwise [4].

2.4. Genetic algorithms

Genetic algorithm (GA) is a search heuristic that imitates the process of natural selection. These algorithms are commonly used for solving optimization problems. GA are a subclass of a larger group of algorithms called evolutionary algorithms(EA) which use techniques inspired by natural evolution(mutation, inheritance, crossover, selection) in order to solve optimization problems. For each of the algorithms that belong to GA class, there are several basic steps:

1. Initialize the population

2. Following the particular algorithm(depending on the fitness value of each candidate), create next generation

3. Repeat the second step until the condition to stop is reached (maximum number of generations is reached or an acceptable solution is derived)

Second step is the most important step for each algorithm and is the only step that differs among different algorithms. For solving S-boxes optimization problems, several genetic algorithms are used within the ECF. All of them use the same fitness function for evaluation of each candidate, only implement different "*advanceGeneration*" method.

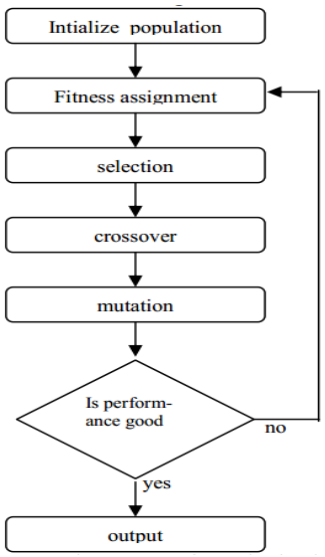

Figure 2.5 Basic structure of genetic algorithm [6]

2.4.1. Steady State Genetic Algorithm

This GA is steady state meaning that there are no generations. It differs from the generic GA because tournament selection does not replace the selected individuals in the population. Instead of adding the children of the selected parents into the next generation, the two best individuals out of the two parents and two children are added back into the population so that the population size remains constant.

```
bool SteadyStateTournament::advanceGeneration(StateP state, DemeP deme)
{
    //! this algorithm: one _generation_ is 'deme->size()' _iterations_
    for(uint iIter = 0; iIter < deme->size(); iIter++) {

        ECF_LOG(state, 5, "Individuals in tournament: ");

        std::vector<IndividualP> tournament;
        for (uint i = 0; i < nTournament_; ++i) {
            // select a random individual for the tournament
            tournament.push_back(selRandomOp->select(*deme));
            ECF_LOG(state, 5, uint2str(i) + ": " + tournament[i]->toString());
        }

        // select the worst from the tournament
        IndividualP worst = selWorstOp->select(tournament);
        ECF_LOG(state, 5, "The worst from the tournament: " + worst->toString());

        // remove pointer to 'worst' individual from vector 'tournament'
        removeFrom(worst, tournament);

        // crossover the first two (remaining) individuals in the tournament
        mate(tournament[0], tournament[1], worst);

        // perform mutation on new individual
        mutate(worst);

        // create new fitness
        evaluate(worst);
        ECF_LOG(state, 5, "New individual: " + worst->toString());
    }

    return true;
}
```

Figure 2.6 NextGeneration method for Steady State Tournament algorithm[ECF]

The algorithm takes in a parameter “tournament size” which determines the number of randomly selected individuals from a population. The minimum number for that parameter is 3. Algorithm randomly selects three individuals from the population. According to their fitness function, it selects the worst one and removes it from the tournament (and also from the population). Next step is crossover of the two remaining individuals. Newly obtained individual is first mutated and then its fitness is calculated using evaluate method.

2.4.2. Fitness proportionate selection or roulette wheel selection(RWS)

RWS is a genetic operator used in genetic algorithms for selecting potentially useful solutions for recombination. In fitness proportionate selection, as in all selection methods, the fitness function assigns a fitness to possible solutions or chromosomes. This fitness level is used to associate a probability of selection with each individual chromosome.

If f_i is the fitness of individual i in the population, its probability of being selected is

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j} \qquad [2.1]$$

where *N* is the number of individuals in the population [2].

```
bool RouletteWheel :: advanceGeneration(StateP state, DemeP deme)
{
    // elitism: copy current best individual
    IndividualP best = selBestOp->select(*deme);
    best = copy(best);

    // select individuals
    std::vector<IndividualP> wheel;
    wheel = selFitPropOp->selectMany(*deme, (uint) deme->size());

    // copy selected to new population
    for(uint i = 0; i < wheel.size(); ++i)
        wheel[i] = copy(wheel[i]);

    // replace old population
    for(uint i = 0; i < deme->size(); i++)
        replaceWith((*deme)[i], wheel[i]);

    ECF_LOG(state, 5, "Selected individuals:");
    for(uint i = 0; i < deme->size(); i++){
        ECF_LOG(state, 5, dbl2str(deme->at(i)->fitness->getValue()));
    }

    // determine the number of crx operations
    uint noCrx = (int)(deme->size() * crxRate_ /2);

    // perform crossover
    for(uint i = 0; i < noCrx; i++){

        // select parents
        IndividualP parent1 = selRandomOp->select(*deme);
        IndividualP parent2 = selRandomOp->select(*deme);
        ECF_LOG(state, 5, "Parents: " + dbl2str(parent1->fitness->getValue()) + ", " + dbl2str(parent2->fitness->getValue()));

        // create children
        IndividualP child1 = copy(parent1);
        IndividualP child2 = copy(parent2);

        // perform crx operations
        mate(parent1, parent2, child1);
        mate(parent1, parent2, child2);

        // replace parents with children
        replaceWith(parent1, child1);
        replaceWith(parent2, child2);
    }
    // perform mutation on whole population
    mutate(*deme);

    // evaluate new individuals
    for(uint i = 0; i < deme->size(); i++)
        if(!deme->at(i)->fitness->isValid()) {
            evaluate(deme->at(i));
        }

    // elitism: preserve best individual
    IndividualP random = selRandomOp->select(*deme);
    if(best->fitness->isBetterThan(random->fitness))
        replaceWith(random, best);

    return true;
}
```

Figure 2.7 NextGeneration method for Roulette Wheel algorithm[ECF]

Best individual from the current population is being selected. Other individuals that form the wheel are selected randomly. New population is being created using individuals from the wheel and old population is being replaced by the new one.

Next step is to calculate the number of crx operations depending on crxRate_ variable which is a parameter of this particular algorithm and is written in configuration file. The variable crxRate_ represents crossover rate.

Depending on the calculated value saved in noCrx variable, we perform crossover in a for loop. From the two randomly selected individuals that represent parents (parent1 and parent) we create two children (child1 and child2). After the children are created, we perform crx operations, i.e., we mate parent1 and parent2 to get both children. In the last step, parents are being replaced by their children in the population.

After replacing given number of individuals by their parents, the whole population is being mutated and evaluated again. Last step is to preserve the best individual – we take a random individual from the population and swap it with the best one we saved at the beginning.

3. Experiments

Using ECF framework and two arbitrary chosen algorithms, we try to derive maximum possible value of fitness. In every experiment, initial parameters are read in from configuration file as explained above.

For all the experiments it has been set that the aim value of FitnessMax variable is 112 – in order for experiment to finish, either this number must be reached or maximum number of generations (set in the configuration file). There is also a third possibility – in the case when, within 50 generations (it is initially set), no significant change happened, algorithm finishes. Furthermore, it is set that the balancedness property is used as a penalty. As mentioned in section 3.3., there are two ways to calculate fitness function of an S-box. In these experiments first method is used.

Steady State Tournament

For all the experiments made by using Steady State Tournament algorithm, the only modifiable parameter (Tournament size) is set to 3.

In table 4.1., results are shown in case when population size is equal to 10. By increasing the maximum number of generations it is possible to get better results. On the other hand, after a certain number, further increasing will not improve the result significantly.

Table 3.1 Results of experiments using SST algorithm, n = 10

Maximum number of generations:	**10**	**50**	**100**	**200**
1.	96.217	98.234	98.252	98.265
2.	96.251	98.237	96.262	96.299
3.	96.216	98.227	96.259	96.295
4.	96.217	98.254	98.279	98.259
5.	96.219	98.226	98.240	98.260
6.	96.208	98.237	96.270	98.268
7.	98.205	96.265	96.265	98.274
8.	96.219	96.246	98.263	98.262
9.	96.226	96.252	98.271	98.252
10.	94.221	98.222	98.254	98.269
Avg:	**96.213**	**97.640**	**97.462**	**97.870**

Therefore, we can increase the number of individuals per generation. By changing that number, we gain more variety and possibly better solution.

Table 3.2. shows the results of the same experiment, but with population size set to 20.

Table 4.2 Results of experiments using SST algorithm, n = 20

Maximum number of generations:	**100**	**200**
1.	98.266	98.281
2.	98.255	98.284
3.	98.250	98.331
4.	98.266	98.285
5.	98.283	98.271
6.	98.258	98.281
7.	98.261	98.306
8.	98.306	98.276
9.	98.276	98.301
10.	98.271	98.273
Avg:	**98.269**	**98.289**

Roulette Wheel

In the case of Roulette Wheel algorithm, default parameters are used – crossover rate is set to 0.5 and selection pressure to 10. As explained above, the experiments are done with the population of 10 and 20 individuals and in a range from 10 to 200 maximum iterations. The results are given in tables 3 and 4, showing the value of FitnessMax.

Table 3.3 Results of experiments using Roulette Wheel algorithm, n = 10

Maximum number of generations:	**10**	**50**	**100**	**200**
1.	96.195	98.215	98.246	98.232
2.	96.206	98.216	98.218	98.249
3.	96.220	98.201	98.221	98.240
4.	98.196	96.243	98.244	98.251
5.	96.198	96.237	98.216	98.251
6.	96.206	96.223	98.207	98.247
7.	96.211	96.238	96.249	98.256
8.	96.211	98.220	98.235	98.226
9.	96.205	98.236	98.227	98.252
10.	96.211	98.217	98.223	98.251
Avg:	**96.406**	**97.425**	**98.029**	**98.246**

Table 3.4 Results of experiments using Roulette Wheel algorithm, n = 20

Maximum number of generations:	**100**	**200**
1.	98.229	98.233
2.	98.206	98.231
3.	98.234	98.252
4.	98.235	98.253
5.	98.218	98.252
6.	98.217	98.245
7.	98.244	98.238
8.	98.245	98.239
9.	98.206	98.262
10.	98.235	98.244
Avg:	**98.227**	**98.245**

Comparison of the upper two algorithms

Above explained two algorithms, derive solution to the problem using different algorithms for creation of new generations. In table 4.5. we can see that, although at the beginning approximately equal, algorithm Roulette Wheel, with increase of generation and/or number of individuals gives better results. Furthermore, experiments with bigger number of max generations and/or with bigger number of individuals, would increase the distance between these two algorithms' solutions.

Table 3.5. Comparison of the results of the two different algorithms, n = 10

Steady State Tournament	**Maximum number of generations:**	**10**	**50**	**100**	**200**
parameters: Number of individuals in a tournament: 3	**FitnessMax value (avg):**	96.213	97.640	97.462	97.870
Roulette Wheel **parameters:** Crossover rate: 0.5 Selection pressure: 10	**Maximum number of generations:**	**10**	**50**	**100**	**200**
	FitnessMax value (avg):	96.406	97.425	98.029	98.246

Table 3.6 Comparison of the results of the two different algorithms, n = 20

Steady State Tournament	**Maximum number of generations:**	**100**	**200**
parameters: Number of individuals in a tournament: 3	**FitnessMax value:**	98.269	98.289
Roulette Wheel **parameters:** Crossover rate: 0.5 Selection pressure: 10	**Maximum number of generations:**	**100**	**200**
	FitnessMax value:	98.227	98.245

4. Future work

4.1. Multi-objective evolutionary algorithms

Multi-objective optimization (also known as Pareto optimization) is an area of multiple criteria decision making, that is concerned with mathematical optimization problems involving more than one objective function to be optimized simultaneously [2].

It found its purpose in many fields of science, particularly where optimal solution needs to be found taking into account the presence of trade-offs between two or more conflicting objectives. For more complicated optimization problems, unique solution that gives an optimum of all objectives simultaneously does not exist. The objective functions are said to be conflicting and there are possibly infinite number of Pareto optimal solutions.

A solution is Pareto optimal if it is impossible to improve one objective function in value without lowering some of the other objective values. All Pareto optimal solutions are considered equally good. Which solution is to be elected and implemented in practice depends on subjective preferences of decision maker(DM). The DM is supposed to be an expert in an optimization problem domain.

Multi-objective optimization methods can be divided into four classes:

1. No preference methods – neutral compromise solution is identified without preference information (no DM).

2. A priori methods - preference information is first asked from the DM and then a solution which satisfies these preferences is found.

3. A posteriori methods – first a representative set of Pareto optimal solutions is found and then the DM must choose one of them.

4. Interactive methods – DM is allowed to iteratively search for the most preferred solution. The DM can stop the search whenever he wants to [2].

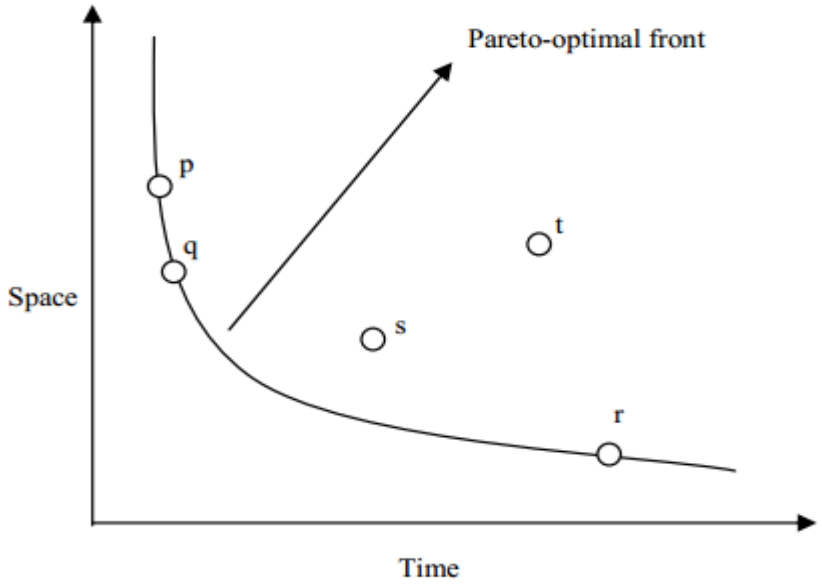

Figure 4.1. Pareto optimal solutions[6]

We can show an example of Pareto optimal solution with the time and space complexity of an algorithm. The problem is to minimize both time and space complexity. The point 'p' represents a solution, which has a minimal time, but the space complexity is high. On the other hand, the point 'r' represents a solution with high time complexity but minimum space complexity. Considering both objectives, no solution is optimal. Therefore, we can't say that solution 'p' is better or worse than 'r'. Which solution is going to be selected, depends on human decision maker and/or his needs.

4.2. Predator-prey algorithm

Predator-prey algorithm is multi-objective evolution strategy. This algorithm does not use a domination check to assign fitness to a solution. Instead, predator-prey concept is used. Preys represent a set of solutions which are placed on the vertices of an undirected connected graph.

First each predator is associated with a particular objective function and randomly placed on any vertex of the graph. After at least *N* predators were initialized in the graph, they look around for preys in their neighborhood vertices.

Each predator catches a prey with worst value of its associated objective function. Every prey that is caught is erased from the vertex and a new solution is obtained by mutating and recombining random prey in the neighborhood. After this step, new individuals are made and predators move to randomly selected neighboring vertices. This procedure ends when specified number of iterations has elapsed. The main advantage of this method is its simplicity – randomly selected graph vertices and replacing worst solutions of individual objectives with mutated solutions. Furthermore, algorithm is sensitive to the choice of mutation strength parameter and the ratio of predator and prey [6]. In the case of AES S-box optimization problem, depending on the number of properties we take into account, number of predators would be chosen – *N* equals to the number of selected evaluated properties. If this algorithm is used in above described optimization problem, so taking into account three properties (*N* = 3), the pseudo-code would be:

while (condition satisfied or maximum number of iteration reached) {

1. Initialize specified number of individuals in initial generation

2. Place all individuals in vertices of the connected graph

3. Create 3 predators and assign each of them its objective function (balancedness, transparency order, nonlinearity)

4. Place predators in randomly selected vertices of the graph

5. Each predator has to find the worst individual in its neighboring vertices depending on assigned objective function

6. Every selected (worst) individual is removed from the graph

7. Each predator now randomly selects another individual from its surrounding and creates a new individual using mutation and recombination and puts it in the place of erased one

8. The procedure is over, every predator is randomly set to a new position (vertex)

}

5. Conclusion

Cryptology is one of the most important fields of computer science due to growing necessity for protection of personal and/or confidential information. New solutions to a big number of attacks on privacy and confidentiality are being investigated and developed each day. As one of the most used algorithms for encryption, AES is constantly trying to improve and find better way to hide the encryption process. Therefore, S-boxes are vital. Furthermore, to develop the best possible S-box, with optimized parameters, which in the end represents the resistance to different attacks and the level of safety it can offer to its clients, evolutionary computation and genetic algorithms have found its use. Instead of randomly adjusting each possible parameter, given the fact that there are often trade-offs between them, they apply the laws of nature (mutation and crossover) to develop the solution with wanted parameters or even better. The standard programming and way of thinking would be useless when trying to solve optimization problem of

S-boxes, even taking into account only 3 properties. Since every of those properties has a value from a certain set of values, increments and decrements are possible for whatever value we want, there is a space of infinite possible solutions. Furthermore, changing only one parameter can affect the whole value drastically, for better or worse. Due to a time limit, it is impossible to find all of the possible solutions, evaluate them and then determine the best one. Therefore, genetic algorithms are here of great help. Using them we might not be able to achieve an optimal solution, but we will find a solution that is good enough much faster and simpler then randomly trying to determine which values are good. In the results of experiments within this paper could not reach even 99. This gives us a reason to try to adjust the parameters within ECF in order to reach higher value of MaxFitness variable (property of an S-box). After every adjustment, by redoing experiments, it will be shown whether new parameters affected it for better or worse. So in order to achieve better results, genetic algorithms and Evolutionary Computation Framework will be of great help.

6. Abstract

Examination of methods for evaluation of cryptographic primitives, with focus on AES S-box. Description of properties of S-boxes and Boolean functions, important for evaluation of AES S-boxes within the Evolutionary Computation Framework (ECF). Experiments description and results, using ECF and implemented algorithms – Roulette Wheel and Steady State Tournament. Comparison of the results given by Roulette Wheel and Steady State Tournament is given. As future work, it is proposed to build new evolutionary algorithm into ECF and try multi objective evolutionary algorithms for S-box optimization.

7. References

[1] Evolving S-Boxes for Improved Resilience to DPA Attacks, 2013., S.Picek

[2] Wikipedia, http://www.wikipedia.com/

[3] Cryptographic properties of Boolean functions and S-boxes, 2006., S.Picek

[4] Evolving DPA-resistant Boolean Functions, S.Picek

[5] http://searchsecurity.techtarget.com/, *Cryptography*, 12.05.2014.

[6] Evolutionary Algorithms for Multi-Criterion Optimization: A Survey, Ashish Ghosh, http://www.ijcis.info/international%20journal%20of%20computing%20and%20information%20sciences_files/vol2n1/38-57s.pdf, 2004.

[7] On the support of the Walsh transforms of Boolean functions, Claude Carlet and Sihem Mesnager, http://eprint.iacr.org/2004/256.pdf