# Cryptography and Cryptanalysis Through Computational Intelligence

E.C. Laskari[1,4], G.C. Meletiou[2,4], Y.C. Stamatiou[3,4], and M.N. Vrahatis[1,4]

[1] Computational Intelligence Laboratory, Department of Mathematics, University of Patras, GR–26110 Patras, Greece
elena@math.upatras.gr, vrahatis@math.upatras.gr

[2] A.T.E.I. of Epirus, Arta, Greece, P.O. Box 110, GR–47100 Arta, Greece
gmelet@teiep.gr

[3] University of Ioannina, Department of Mathematics, GR–45110 Ioannina, Greece istamat@uoi.gr

[4] University of Patras Artificial Intelligence Research Center (UPAIRC), University of Patras, GR–26110 Patras, Greece

The past decade has witnessed an increasing interest in the application of Computational Intelligence methods to problems derived from the field of cryptography and cryptanalysis. This phenomenon can be attributed both to the effectiveness of these methods to handle hard problems, and to the major importance of automated techniques in the design and cryptanalysis of cryptosystems.

This chapter begins with a brief introduction to cryptography and Computational Intelligence methods. A short survey of the applications of Computational Intelligence to cryptographic problems follows, and our contribution in this field is presented. Specifically, some cryptographic problems are viewed as discrete optimization tasks and Evolutionary Computation methods are utilized to address them. Furthermore, the effectiveness of Artificial Neural Networks to approximate some cryptographic functions is studied. Finally, theoretical issues of Ridge Polynomial Networks and cryptography are presented.

The experimental results reported suggest that problem formulation and representation are critical determinants of the performance of Computational Intelligence methods in cryptography. Moreover, since strong cryptosystems should not reveal any patterns of the encrypted messages or their inner structure, it appears that Computational Intelligence methods can constitute a first measure of the cryptosystems' security.

## 1.1 Introduction

A basic task of cryptography is the transformation, or *encryption*, of a given message into another message which appears meaningful only to the intended recipient through the process of *decryption*. The message that undergoes encryption is called the *plaintext* (or cleartext), while the transformed message is called *ciphertext*. *Cryptanalysis* refers to the process of discovering the plaintext from the ciphertext without knowing the decryption key. A cryptographic algorithm, *cipher*, is a mathematical function employed for the encryption and decryption of messages. Ciphers can be divided into two categories, the *symmetric-key* and the *public-key* ciphers. In symmetric-key ciphers the sender and the receiver of the message secretly choose the key that will be used for encryption and decryption. A drawback of this type of cryptosystems is that it requires prior communication of the key between the sender and the receiver, through a secure channel, before any message is sent.

Public-key ciphers are designed in such a way that the key used for encryption is publicly available and differs from the key used in decryption, which is secret. Although these two keys are functionally interrelated, the computation of the secret key from the public key is computationally intractable. Thus, using the public key anyone can send an encrypted message, but only the owner of the secret key can perform the decryption. Next, we briefly present the cryptographic algorithms that are used in the reported cryptanalysis experiments.

### 1.1.1 Block ciphers

A block cipher is a function which maps $n$-bit plaintext blocks to $n$-bit ciphertext blocks, where $n$ is a chosen blocklength. The function is parameterized by a $k$-bit key $K$, which takes values from a subset $\mathcal{K}$, the *key space*, of the set of all $k$-bit vectors. The function must be invertible to allow unique decryption. Block ciphers can be either symmetric-key or public-key [85].

A *Feistel cipher* is an iterated block cipher based on the repetitive computation of simple functions, called *round functions*, on the input data, for a predetermined number of rounds. The resulting function maps an $n$–bit plaintext $P$, to a ciphertext $C$. In a Feistel cipher the currently computed (by the round function) $n$–bit word is divided into $(n/2)$–bit parts, the left part $L_i$ and the right part $R_i$ [32]. Then, the $i$th round, $1 \leqslant i \leqslant r$, has the following effect:

$$L_i = R_{i-1}, \quad R_i = L_{i-1} \oplus F_i(R_{i-1}, K_i), \tag{1.1}$$

where $K_i$ is the subkey used in the $i$th round (derived from the cipher key $K$), and $F_i$ is an arbitrary round function for the $i$th round. After the last round function has been performed, the two halves are swapped and the outcome is the ciphertext $C$ of the Feistel cipher, i.e. $C = (R_r, L_r)$. The encryption procedure of Feistel ciphers is illustrated in Fig. 1.1.

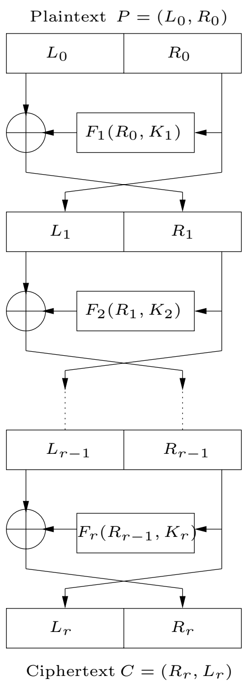

**Fig. 1.1.** The encryption procedure of Feistel ciphers

On Feistel based cryptosystems the decryption function is simply derived from the encryption function by applying the subkeys, $K_i$, and the round functions, $F_i$, in reverse order. This renders the Feistel structure an attractive choice for software and hardware implementations.

One of the most widely used (for non-critical applications) Feistel ciphers is the Data Encryption Standard (DES) [94]. DES is a symmetric-key cryptosystem, meaning that the parties exchanging information possess the same key. It processes plaintext blocks of $n = 64$ bits, producing 64–bit ciphertext blocks. The effective key size is $k = 64$ bits, 8 of which can be used as parity bits. The main part of the round function is the $F$ function, which works on the right half of the data, using a subkey of 48 bits and eight S-boxes. *S-boxes* are mappings that transform 6 bits into 4 bits in a nonlinear manner and constitute the only nonlinear component of DES. The 32 output bits of the $F$ function are XORed with the left half of the data and the two halves are subsequently exchanged. A detailed description of the DES algorithm can be found in [85, 125].

Two of the most powerful cryptanalytic attacks for Feistel based ciphers, rely on the exploitation of specific weaknesses of the S-boxes of the target cryptoalgorithm. These attacks are the *Linear Cryptanalysis* [78, 79] and the *Differential Cryptanalysis* [6, 7], which were successfully applied first to the cryptanalysis of DES. Differential Cryptanalysis (DC) is a *chosen plaintext* attack. In chosen plaintext attacks the opponent has temporary access to the encryption function and thus he/she can choose some plaintexts and construct the corresponding ciphertexts. DC analyzes the effect of particular differences in plaintext pairs on the differences of the resulting ciphertext pairs. These differences can be used to assign probabilities to the possible keys and to identify bits of the key that was used in the encryption process. This method usually works on a number of pairs of plaintexts having a specific difference and relies on the resulting ciphertext pairs only. For cryptosystems similar to DES, the difference is chosen to be a fixed XORed value of the two plaintexts.

To locate the most probable key, DC employs *characteristics*. Note that any pair of encrypted plaintexts is associated with the XOR value of its two plaintexts, the XOR value of its ciphertexts, the XOR values of the inputs of each round in the two encryption executions and the XOR values of the outputs of each round in the two encryption executions. These XOR values form an $r$-round characteristic [6]. More formally, an *$r$-round characteristic* is a tuple $\Omega = (\Omega_P, \Omega_\Lambda, \Omega_C)$, where $\Omega_P$ and $\Omega_C$ are $n$ bit numbers and $\Omega_\Lambda$ is a list of $r$ elements $\Omega_\Lambda = (\Lambda_1, \Lambda_2, \ldots, \Lambda_r)$, each of which is a pair of the form $\Lambda_i = (\lambda_I^i, \lambda_O^i)$, where $\lambda_I^i$ and $\lambda_O^i$ are $n/2$ bit numbers and $n$ is the block size of the cryptosystem [6]. A characteristic satisfies the following requirements:

(a) $\lambda_I^1$ is the right half of $\Omega_P$,
(b) $\lambda_I^2$ is the left half of $\Omega_P \oplus \lambda_O^1$,
(c) $\lambda_I^r$ is the right half of $\Omega_C$,
(d) $\lambda_I^{r-1}$ is the left half of $\Omega_C \oplus \lambda_O^r$, and
(e) for every $i$, $2 \leqslant i \leqslant r-1$, it holds that $\lambda_O^i = \lambda_I^{i-1} \oplus \lambda_I^{i+1}$.

To each characteristic is assigned the probability of a random pair with the chosen plaintext XOR, $\Omega_P$, having the round XOR values, $\Lambda_i$, and the ciphertext XOR, $\Omega_C$, specified in the characteristic. Each characteristic allows the search for a particular set of bits in the subkey of the last round: the bits that enter particular S-boxes, depending on the chosen characteristic. The characteristics that are most useful are those that have a maximal probability and a maximal number of subkey bits whose occurrences can be counted.

DC is a statistical method that rarely fails. A more extended analysis on DC and its results on DES for different numbers of rounds is provided in [6]. DC was the first theoretical cryptanalysis for DES requiring (on average) less steps than the brute force attack, i.e., testing all $2^{56} = 72\ 057\ 594\ 037\ 927\ 936$ possible keys. Although this number appears prohibitive, a brute-force attack on 56-bit DES, using technology standards of previous decades, has been successfully launched. A specially designed hardware with appropriate software, designed and built by the Cryptography Research, the Advanced Wireless Technologies, and the EFF (Electronic Frontier Foundation) reached a rate of key searches of about 90 billion keys per second. Their prototype, called Deep Crack, contains 29 boards each containing 64 specially designed chips. The achieved key search led to the determination of the key in the RSA DES challenge ($10.000 worth) after an, approximately, 56 hours search effort on July 15 in 1998. Moreover, total cost remained at a relatively low level, below $250.000 (much more lower today) which renders their achievement even more important and worrying as far as the security of 56-bit DES is concerned. However, as if anticipating this attack, the National Institute of Standards and Technology (NIST) had already initiated in 1997 an international contest, accepting proposals for what will become the new standard to replace DES. The contest winner was called *Advanced Encryption Standard* (AES) and it is expected to withstand attacks for a period of at least 30 years, as Miles Smid, the manager of the security technology division of NIST, stated. AES became the government standard and it is also used by private companies (on a royalty-free basis). In 1998, NIST announced the acceptance of fifteen candidate algorithms (in the first round of the process) and resorted to the cryptography community to investigate their security and efficiency. After reviewing the studies and the relevant reports, NIST selected five finalists (Round 2). Among these Rijndael, proposed by Daemen and Rijmen was selected as the new DES. The other four finalists were: MARS (proposed by IBM), RC6 (proposed by RSA Laboratories), Serpent by Anderson, Biham, and Knudsen and Twofish by Schneier, Kelsey, Whiting, Wagner, Hall, and Ferguson. In a third round, NIST concluded that the *Rijndael* cipher should become the Advanced Encryption Standard. Since then, various attacks have been proposed on this cipher but none with devastating effects.

In Sect. 1.4.2 the problem of finding some missing bits of the key that is used in a simple Feistel cipher, namely the Data Encryption Standard with four and six rounds, respectively is studied.

### 1.1.2 Public key cryptographic schemes

Public key cryptography is intimately related to a number of hard and complex mathematical problems from the fields of computational algebra, number theory, probability theory, mathematical logic, Diophantine's complexity and algebraic geometry. Such problems are the factorization [112], the discrete logarithm [1, 96, 104] and others [86]. Cryptosystems rely on the assumption that these problems are computationally intractable, in the sense that their computation cannot be completed in polynomial time.

**Discrete Logarithm Problem (DLP):** DLP amounts to the development of an efficient algorithm for the computation of an integer $x$ that satisfies the relation $\alpha^x = \beta$, where $\alpha$ is a fixed primitive element of a finite field $\mathbb{F}_q$ (i.e., $\alpha$ is a generator of the multiplicative group $\mathbb{F}_q^*$ of $\mathbb{F}_q$) and $\beta$ is a non–zero element of the field. We assume that $x$ is the smallest nonnegative integer with $\alpha^x = \beta$. Then, $x$ is called the *index*, or the *discrete logarithm*, of $\beta$. In the special case of a finite field $\mathbb{Z}_p$ of prime order $p$, a primitive root $g$ modulo $p$ is selected. If $u$ is the smallest nonnegative integer with

$$g^u \equiv h \pmod{p}, \tag{1.2}$$

then $u$ is called the *index*, or the *discrete logarithm*, of $h$ [1, 96, 104].

The security of various public and symmetric key cryptosystems [1, 22, 29, 82, 83, 93, 95, 96, 104, 133], namely the *Diffie–Hellman exchange protocol* [25], the *El Gamal public key cryptosystem*, as well as, *the El Gamal digital signature scheme* [29], relies on the assumption that DLP is computationally intractable.

**Diffie–Hellman key Problem (DHP):** DHP is defined as follows [22, 80, 133]. Let $\alpha$ be a fixed primitive element of a finite field $\mathbb{F}_q$; $x$, $y$, satisfying, $0 \leqslant x, y \leqslant q-2$, denote the private keys of two users; and $\beta = \alpha^x$, $\gamma = \alpha^y$ represent the corresponding public keys. Then, the problem amounts to computing $\alpha^{xy}$ from $\beta$ and $\gamma$, where $\alpha^{xy}$ is the symmetric key for secret communication between the two users. Consider the special case of the DHP, where $\beta = \gamma$. The term *Diffie–Hellman Mapping* refers to the mapping,

$$\beta = \alpha^x \longmapsto \alpha^{x^2}. \tag{1.3}$$

**Diffie–Hellman Mapping Problem (DHMP):** The definition of the DHMP follows naturally from the aforementioned definition of DHP. The two problems, DHP and DHMP, are computationally equivalent, as the following relation holds $\alpha^{x^2}\alpha^{y^2}\alpha^{2xy} = \alpha^{(x+y)^2}$, and the computation of $\alpha^{xy}$ from $\alpha^{2xy}$ is feasible (square roots over finite fields).

For the discrete logarithm problem and the Diffie–Hellman key problem, the following theorem holds:

**Theorem 1.** *Let $G$ be a cyclic group of order $m$ and $G = \langle\alpha\rangle$, where $\alpha^m = e$, and $e$ is the neutral element of the group. Then the well–known cryptosystems of DLP and DHP based on the group $G$ can be represented by matrices of the form*

$$\mathbf{x}^\top W \mathbf{y}, \tag{1.4}$$

*where $\mathbf{x}, \mathbf{y}$ are vectors and $W = \{w_{ij}\}_{i,j=1}^{m} = \alpha^{-ij}$.*

*Proof.* The proof follows by taking into consideration that there exists a prime $p$ such that $m|p-1$ (or $m|p^n - 1 = q - 1$). Then $G$ can be considered as a subgroup of the multiplicative group of $\mathbb{Z}_p^*$ (or $\mathrm{GF}^*(p,n)$) and according to [22, 68, 81, 93], such a representation exists.

**Factorization problem:** The factorization problem on the other hand, is related to the RSA cryptosystem and its variants [112]. The security of this cryptosystem relies on the computational intractability of the factorization of a positive integer $N = p \times q$, where $p$ and $q$ are distinct odd primes [85]. The factorization of $N$ is equivalent to determining $\phi(N)$ from $N$, where $\phi(N) = (p-1) \times (q-1)$ [112]. Numerous techniques, including algebraic, number theoretic, soft computing and interpolation methods, have been proposed to tackle the aforementioned problems [1, 22, 62, 96, 120].

In Sects. 1.4.1 and 1.4.3, the DLP, the DHP, the DHMP and the factorization problem are studied in different settings utilizing Evolutionary Computation methods and Artificial Neural Networks, respectively.

### 1.1.3 Elliptic Curve based cryptosystems

Cryptographic systems based on elliptic curves were proposed in [57,90] as an alternative to conventional public key cryptosystems. Their main advantage is the use smaller parameters (in terms of bits) compared to the conventional cryptosystems (e.g. RSA). This is due to the apparently increased difficulty of the *Elliptic Curve Discrete Logarithm Problem* (ECDLP), which constitutes the underlying mathematical problem. ECDLP is believed to require more time to solve than the time required for the solution of its finite field analogue, the Discrete Logarithm Problem (DLP). The security of cryptosystems that rely on discrete logarithms, relies on the hypothesis these problems cannot be solved in polynomial time. Numerous techniques that exploit algebraic and number theoretic methods, software oriented methods, as well as, approximation and interpolation techniques [22, 67, 80, 83, 134], have been proposed to speed up the solution of these two types of the discrete logarithm problem.

An *Elliptic Curve* over a prime finite field $\mathbb{F}_p$, where $p > 3$ and prime, is denoted by $E(\mathbb{F}_p)$ and is defined as the set of all pairs $(x, y) \in \mathbb{F}_p$ (points in affine coordinates) that satisfy the equation $y^2 = x^3 + ax + b$, where $a, b \in \mathbb{F}_p$, with the restriction $4a^3 + 27b^2 \neq 0$. These points, together with a special point denoted by $\mathcal{O}$, called *point at infinity*, and an appropriately defined point addition operation form an Abelian group. This is the *Elliptic Curve group* and the point $\mathcal{O}$ is its identity element (see [8, 121] for more details on this group). The *order $m$ of an elliptic curve* is defined as the number of points in $E(\mathbb{F}_p)$. According to Hasse's theorem (see e.g., [8,121]) it holds that $p + 1 - 2\sqrt{p} \leqslant m \leqslant p + 1 + 2\sqrt{p}$. The *order of a point $P \in E(\mathbb{F}_p)$* is the smallest positive integer, $n$, for which $nP = \mathcal{O}$. From Lagrange's theorem, it holds that the order of a point is a divisor of the order of the elliptic curve.

DLP can be described as follows. Let $G$ be any group and $h$ one of its elements. Then, the DLP for $G$ to the base $g \in G$ consists of determining an integer, $u$, such that $g^u = h$, when the group operation is written as multiplication, or, $ug = h$ when the group operation is written as addition. In groups formed by elliptic curve points the group operation is addition. Therefore, let $E$ be an elliptic curve over a finite field $\mathbb{F}_q$, $P$ a point on $E(\mathbb{F}_q)$ of order $n$, and $Q$ a point on $E(\mathbb{F}_q)$, such that $Q = tP$, with $0 \leqslant t \leqslant (n-1)$. The ECDLP amounts to determining the value of $t$. The best algorithms to solve the ECDLP require an exponential number of expected steps, in contrast to the best algorithms known today for the DLP defined over the multiplicative group of $\mathbb{F}_q$, which require sub-exponential time in the size of the used group. In Sect. 1.4.4 the problem of computing the least significant bit of the ECDLP using Artificial Neural Netwroks is studied with interesting results.

## 1.2 Computational Intelligence Background and Methods

Alan Turing is considered to be the first who conceived the idea of Artificial and Computational Intelligence as early as 1950, when he hypothesized that computers that mimic the processes of the human brain can be developed. This hypothesis implies that any reasoning can be carried out on a large enough deterministic computer. Turing's hypothesis remains a vision, but it has inspired a great amount of research in the effort to embed intelligence to computers.

Although there is no commonly accepted definition, Computational Intelligence (CI) can be considered as the study of adaptive mechanisms that enable intelligent behavior of a system in complex and changing environments [28,30]. These mechanisms exhibit the ability to learn, or adapt to new situations, such that one or more attributes of reason, such as generalization, discovery, association and abstraction, are perceived to be possessed by the system. To enable intelligent behavior, CI systems are often designed to model aspects of biological and natural intelligence. Thus, CI systems are usually hybrids of paradigms such as Evolutionary Computation systems, Artificial Neural Networks and Fuzzy systems, supplemented with elements of reasoning.

### 1.2.1 Evolutionary Computation

Evolutionary Computation (EC) is a branch of CI that draws its inspiration from evolutionary mechanisms such as natural selection and adaptive behavior, to design optimization and classification methods. Natural selection refers to the survival of the fittest through reproduction. An offspring must retain those characteristics of its parents that are best suited to survive in a given environment. Offsprings that are weak lose the battle of survival. The EC paradigms that form this class are *Genetic Algorithms* (GA), *Genetic Programming* (GP), *Evolutionary Programming* (EP), *Evolution Strategies* (ES) and *Differential Evolution* (DE). The social and adaptive behavior of animals organized in groups inspired the development of another class of EC methods, namely *Swarm Intelligence* (SI). These methods model the social procedures of living organisms that are organized into groups and act for a common cause. Typical examples are the search for food mechanisms of bird flocks, fish schools and ant colonies. The study of many biological processes of social and adaptive behavior led to the opinion that social sharing of information among the individuals of a population can generate an evolutionary advantage [28]. Paradigms of EC that belong to this class of methods are the *Particle Swarm Optimization* (PSO) method and the *Ant Colony Optimization* (ACO) method. In the following sections a brief description of each paradigm of EC is given. Since, the PSO and DE methods will be used in our experiments in Sect. 1.4, they are more thoroughly described.

### Genetic Algorithms

The experimentation of biologists in simulating natural genetic systems using computers gave rise to Genetic Algorithms (GA). John Holland is regarded as the creator of the field of GAs. He studied machine intelligence and machine learning and developed the abilities of GAs to artificial systems [45]. These systems had the ability to adapt to changes of the environment and also exhibited self–adaptation in the sense that they could adjust their operations according to their interaction with the environment. Among the innovations of Holland was the use of a population of individuals for the search procedure instead of a single search point.

The basic concepts of GAs is natural evolution and genetic inheritance. In natural evolution each biological specie has to search for the most appropriate adaptations to a complex and changing environment to ensure its survival. GAs are based on the idea that the knowledge and experience that a specie gains passes in the chromosomes of its members. For this reason the vocabulary used for GAs is that of genetics. Thus, the individuals of the population are called *chromosomes* or *genotypes*. Each chromosome consists of parts called genes and each of them is responsible for the inheritance of one or more characteristics. The evolution procedure of a population of chromosomes corresponds to a search on the space of possible problem solutions and has to balance between two different scopes, the *exploitation* of the best solutions and the *exploration* of the search space. The evolution procedure of GAs is implemented using two operators, *crossover* and *mutation*. These operators alter chromosomes to produce better ones. The selection of the new population is completed using as a criterion a fitness measure. Regarding the representation of the chromosomes, GAs usually employ binary representation, but GA methods that use other arithmetic systems, including floating point numbers, have also been developed. GAs have been successfully applied to optimization problems arising in different fields such as applied mechanics and design, time–scheduling, the traveling salesman's problem, optimal control and robotics, and economics among others [3, 24, 34, 38, 87].

### Evolutionary Programming

The field of Evolutionary Programming (EP) was developed by Larry Fogel [35] parallel to that of GAs. The aim of EP was the evolution of Artificial Intelligence by predicting the changes of the environment. The environment in EP is described as a sequence of symbols from a finite set and the evolution algorithm provides as output a new symbol. This symbol has to maximize the fitness function that is used as a measure for the accuracy of the prediction. For the representation of each individual of the population finite state machines were chosen. Evolutionary Programming, just like GAs, uses the principle of the selection of the fittest for the new population, but only the mutation operator is used for altering the individuals of the population. To this initial version of EP two more basic concepts have been added. The first regards the ability of handling continuous parameters in addition to the discrete ones, and the second is the ability of self–adaptation. Using these new advances, EP can address optimization and classification problems with applications in several scientific fields as for example economics [33, 34].

## Evolution Strategies

In the 1970s, Ingo Rechenberg and Hans-Paul Schwefel used the idea of mutation trying to obtain the optimal design for a sequence of joints in a liquid transition pipe. The classical optimization techniques that make use of the gradient of the fitness function were unable to handle the problem and the only solution was the experimentation with mutation. Using mutation they caused a small perturbation to the best existing problem solutions in order to explore in a stochastic manner the neighborhoods in the search space of the problem. This experimentation was the beginning of the development of Evolution Strategies, which were established in 1973 [109]. Evolution Strategies can be considered as evolutionary programs that use floating point representation and employ a *recombination* and a mutation operator. They have been used for the solution of several optimization problems with continuously changing parameters and they have been recently extended for discrete problems [42].

## Genetic Programming

Genetic Programming (GP) was developed more recently by Koza [61]. The idea behind GP is the following: instead of constructing an evolutionary program to solve the problem, to locate in the space of computational programs the most proper one for the specific case. GP provides means to achieve this goal. A population of executable computational programs is created and every individual program competes with the rest. The non efficient programs become idle while the best ones reproduce by means of operators such as crossover and mutation. The evaluation of the programs is done using a fitness on a predefined set of problems.

## Differential Evolution

The Differential Evolution algorithm (DE) [126] is a parallel direct numerical search method, that utilizes $N$, $D$–dimensional parameter vectors $x_{i,G}$, $i = 1, 2, \ldots, N$, as a population for each iteration (generation) of the algorithm. At each generation, the *mutation* and *crossover* (*recombination* [103, 127]) operators are applied on the individuals, to produce a new population, which is subsequently subjected to the selection phase.

For each vector $x_{i,G}$, $i = 1, 2, \ldots, N$, a *mutant vector* is generated through the following equation:

$$v_{i,G+1} = x_{r_1,G} + F\ (x_{r_2,G} - x_{r_3,G}), \tag{1.5}$$

where $r_1, r_2, r_3 \in \{1, 2, \ldots, N\}$, are random indexes, mutually different and different from $i$, and $F \in (0, 2]$. Consequently, $N$ must be greater than, or equal to, 4. Following the mutation phase, the crossover operator is applied on the mutant vector yielding the *trial vector*, $u_{i,G+1} = (u_{1i,G+1}, u_{2i,G+1}, \ldots, u_{Di,G+1})$, where,

$$u_{ji,G+1} = \begin{cases} v_{ji,G+1}, & \text{if}\,(\text{randb}(j) \leqslant CR) \text{ or } j = \text{rnbr}(i), \\ x_{ji,G}, & \text{if}\,(\text{randb}(j) > CR) \text{ and } j \neq \text{rnbr}(i), \end{cases} \tag{1.6}$$

for $j = 1, 2, \ldots, D$; where randb$(j)$, is the $j$th evaluation of a uniform random number generator in the range $[0, 1]$; *CR* is the (user specified) crossover constant in the range $[0, 1]$; and rnbr$(i)$ is a randomly chosen index from the set $\{1, 2, \ldots, D\}$. To decide whether or not the vector $u_{i,G+1}$ will be a member of the population of the next generation, it is compared to the initial vector $x_{i,G}$. Thus,

$$x_{i,G+1} = \begin{cases} u_{i,G+1}, & \text{if } \ f(u_{i,G+1}) < f(x_{i,G}), \\ x_{i,G}, & \text{otherwise.} \end{cases} \tag{1.7}$$

The DE algorithm that utilizes the mutation operator of (1.5) is called the standard variant of the DE algorithm. Different mutation operators define the other variants of the DE algortihm. The mutation operators that have been applied with promising results [126], are the following:

$$v_{i,G+1} = x_{\text{best},G} + F(x_{r_1,G} - x_{r_2,G}), \tag{1.8}$$
$$v_{i,G+1} = x_{i,G} + F(x_{\text{best},G} - x_{i,G}) + F(x_{r_1,G} - x_{r_2,G}), \tag{1.9}$$
$$v_{i,G+1} = x_{\text{best},G} + F(x_{r_1,G} + x_{r_2,G} - x_{r_3,G} - x_{r_4,G}), \tag{1.10}$$
$$v_{i,G+1} = x_{r_1,G} + F(x_{r_2,G} + x_{r_3,G} - x_{r_4,G} - x_{r_5,G}), \tag{1.11}$$

where, $x_{\text{best},G}$, corresponds to the best individual of the $G$th generation, $r_1$, $r_2$, $r_3$, $r_4$, $r_5 \in \{1, 2, \ldots, N\}$, are mutually different random indexes and $x_{i,G}$ is the current individual of generation $G$.

### Particle Swarm Optimization

Particle Swarm Optimization (PSO) method is a population–based algorithm that exploits a population of individuals, to identify promising regions of the search space. In this context, the population is called *swarm* and the individuals are called *particles*. Each particle moves with an adaptable velocity within the search space, and retains in its memory the best position it ever encountered. In the *global* variant of the PSO the best position ever attained by all individuals of the swarm is communicated to all the particles. In the *local* variant, each particle is assigned to a neighborhood consisting of a prespecified number of particles. In this case, the best position ever attained by the particles that comprise the neighborhood is communicated among them [28].

Assume a $D$–dimensional search space, $\mathbf{S} \subset \mathbb{R}^D$, and a swarm of $N$ particles. The $i$th particle is in effect a $D$–dimensional vector $X_i = (x_{i1}, x_{i2}, \ldots, x_{iD})^\top$. The *velocity* of this particle is also a $D$–dimensional vector, $V_i = (v_{i1}, v_{i2}, \ldots, v_{iD})^\top$. The *best previous position* ever encountered by the $i$–th particle is a point in $\mathbf{S}$, denoted by $P_i = (p_{i1}, p_{i2}, \ldots, p_{iD})^\top$. Assume $g$, to be the index of the particle that attained the best previous position among all the individuals of the swarm (global variant of PSO) or among all individuals of the neighborhood of the $i$-th particle (local variant of PSO).

Then, according to the *constriction factor* version of PSO the swarm is manipulated using the following equations [21]:

$$V_i^{(t+1)} = \chi \left( V_i^{(t)} + c_1 r_1 \left( P_i^{(t)} - X_i^{(t)} \right) + c_2 r_2 \left( P_g^{(t)} - X_i^{(t)} \right) \right), \quad (1.12)$$

$$X_i^{(t+1)} = X_i^{(t)} + V_i^{(t+1)}, \quad (1.13)$$

where $i = 1, 2, \ldots, N$; $\chi$ is the constriction factor; $c_1$ and $c_2$ denote the *cognitive* and *social* parameters respectively; $r_1$, $r_2$ are random numbers uniformly distributed in the range $[0, 1]$; and $t$, stands for the counter of iterations. The value of the constriction factor is typically obtained through the formula $\chi = 2\kappa / |2 - \varphi - \sqrt{\varphi^2 - 4\varphi}|$, for $\varphi > 4$, where $\varphi = c_1 + c_2$, and $\kappa = 1$. The default parameter values found in the literature [21] are $\chi = 0.729$ and $c_1 = c_2 = 2.05$. Different configurations of $\chi$ as well as a theoretical analysis of the derivation of the above formula can be found in [21].

In a different version of PSO a parameter called *inertia weight* is used, and the swarm is manipulated according to the formulae [28, 52, 118]:

$$V_i^{(t+1)} = w V_i^{(t)} + c_1 r_1 \left( P_i^{(t)} - X_i^{(t)} \right) + c_2 r_2 \left( P_g^{(t)} - X_i^{(t)} \right), \quad (1.14)$$

$$X_i^{(t+1)} = X_i^{(t)} + V_i^{(t+1)}, \quad (1.15)$$

where $i = 1, 2, \ldots, N$; and $w$ is the inertia weight, while all other variables are the same as in the constriction factor version. There is no explicit formula for the determination of the factor $w$, which controls the impact of the previous history of velocities on the current one. However, since a large inertia weight facilitates global exploration (searching new areas), while a small one tends to facilitate local exploration (fine–tuning the current search area), it appears intuitively appealing to initially set it to a large value and gradually decrease it to obtain more refined solutions. The superiority of this approach against the selection of a constant inertia weight, has been experimentally verified [118]. An initial value around 1.2 and a gradual decline toward 0.1 is considered a good choice for $w$. Proper fine–tuning of the parameters $c_1$ and $c_2$, results in faster convergence and alleviation of local minima. As default values, $c_1 = c_2 = 2$ have been proposed, but experimental results indicate that alternative configurations, depending on the problem at hand, can produce superior performance [52, 98].

In order to avoid velocities from assuming large values that lead to fluctuation of the particles over the search space, and destroy the dynamic of the method, a maximum value, $V_{\max}$, is set for each coordinate of the velocity.

Typically, the swarm and the velocities, are initialized randomly in the search space. For more sophisticated techniques, see [97, 99]. The performance of the PSO method for the Integer Programming problem and the Minimax problem was studied in [71, 72], respectively, with very promising results.

### Ant Colony Optimization

The Ant Colony Optimization (ACO) algorithm is a Swarm Intelligence method for tackling, in general, Combinatorial Optimization problems, like the traveling salesman problem and telecommunications scheduling. It exploits a population of members called *artificial ants* and it has been inspired from experiments with real ant colonies. In these experiments it was discovered that after a small time interval groups of ants choose the shortest between two routes to transfer food to their nest. This ability becomes possible by a chemical substance, called *pheromone*, which ants leave in the environment, that serves as an indirect communication mechanism. Thus, at the beginning the route chosen by ants appears to be random but with the progress of time the possibility of choosing the shortest path becomes higher as the quantity of pheromone on this path increases faster compared to the quantity of pheromone on longer paths. This simple idea is implemented by the ACO methods to locate solutions and address hard optimization problems [10, 27].

## 1.2.2 Artificial Neural Networks

The complex and parallel functionality of the human brain has motivated the design of *Artificial Neural Networks* (ANNs). An ANN can be considered as a massively parallel distributed processor, comprised of simple units called *neurons*, and characterized by an inherent ability to acquire knowledge from data through a learning process. Knowledge is stored at the interneuron connection strengths, called *weights*, making it thus available for use [41]. Each artificial neuron implements a local computation. The output of this computation is determined by the neuron's input and its activation function. The overall functionality of a network is determined by its *topology* (architecture), i.e. the number of neurons and their interconnection pattern, the training algorithm applied, and its neuron characteristics [46, 102].

ANNs can be categorized based on their topology, their functionality, their training methods, and other characteristics. Regarding their topology, the most simple ANNs have only one layer of neurons and are called single-layer ANNs, while the ones with more than one layers of neurons are called multilayer ANNs. Furthermore, ANNs with acyclic interneuron connections are called Feedforward Neural Networks (FNNs), while those with feedback loops are called Recurrent Neural Networks (RNNs). The most commonly used

ANNs are FNNs. A Feedforward Neural Network is a network with acyclic and one-way directed interneuron connections, where neurons can be grouped into layers. Thus, the network's topology can be described by a series of integers each representing the number of units that belong to the corresponding layer.

The functionality of ANNs is based on the type of neurons they consist of, and their activation function. In general, the are two types of neurons, summing and product neurons. Summing neurons apply their activation function over the sum of the weighted inputs, while product neurons apply their activation function over the product of the weighted inputs (see Sect. 1.5). The activation function determines the output of the neuron, and several types of activation functions can be used. The most commonly encountered ones are the *linear function* (1.16), the *threshold function* (1.17), the *sigmoid function* (1.18), the *hyperbolic tangent function* (1.19) and the *Gaussian function* (1.20).

$$f_1(x) = \alpha x, \tag{1.16}$$

$$f_2(x) = \begin{cases} \alpha_1, & \text{if } x \geqslant \theta, \\ \alpha_2, & \text{if } x < \theta, \end{cases} \tag{1.17}$$

$$f_3(x) = \frac{1}{1 + e^{-\lambda_1 x}}, \tag{1.18}$$

$$f_4(x) = \tanh(\lambda_2 x), \tag{1.19}$$

$$f_5(x) = e^{-x^2/\sigma^2}, \tag{1.20}$$

where $\alpha, \alpha_1, \alpha_2, \theta, \lambda_1, \lambda_2$ are constants and $\sigma^2$ is the variance of the Gaussian distribution. The training methods for ANNs can be divided into three categories, *supervised learning* methods in which case the ANNs must adapt to given data so as to produce a specific output; *unsupervised learning* where ANNs have to discover patterns on the input data; and *reinforcement learning* that aims at rewarding ANNs for good performance and penalize them otherwise [30, 39, 41, 58].

In the case of supervised learning, the goal of training is to assign to the weights (free parameters) of the network, $W$, values such that the difference between the desired output (target) and the actual output of the network is minimized. The adaptation process starts by presenting to the network a series of patterns for which the desired outputs are a priori known, and computing a total error function $E = \sum_{k=1}^{P} E_k$. In this equation, $P$ is the number of patterns and $E_k$ is the partial network error with respect to the $k$th pattern. For the computation of the partial network error a variety of error (distance) functions can be used [74, 132]. Usually, it is computed by summing the squared difference between the actual network outputs and the desired outputs for the corresponding pattern. The training patterns can be presented numerous times to the network. Each pass of all the patterns that belong to the training set, $T$, is called a *training epoch.* The total number of epochs required can be considered as the speed of the training algorithm. Several training algorithms can be found in [41, 73, 75, 76, 103, 110, 130].

The computational power of neural networks derives from their parallel and distributed structure and their inherent ability to adapt to specific problems, learn, and generalize. These characteristics allow ANNs to solve complex problems. In [46,131] the following statement has been proved: *"Standard feedforward networks with only a single hidden layer can approximate any continuous function uniformly on any compact set and any measurable function to any desired degree of accuracy"*. It has also been proved [102] that *"a single hidden layer feedforward network with a fixed number of units in the hidden layer, has a lower bound on the degree of the approximation of any function"*. The lower bound obstacle can be alleviated if more than one hidden layers are used. Mairov and Pincus in [102] have proved that, *"on the unit cube in $\mathbb{R}^n$ any continuous function can be uniformly approximated, to within any error by using a two hidden layer network having $2n + 1$ units in the first layer and $4n+3$ units in the second"*. Furthermore, Anthony in [2] has proved that *"there is a 2-layer threshold network capable of computing any Boolean function"*. These results imply that any lack of success in applications can be attributed to inadequate training, an insufficient number of hidden units, or the lack of a deterministic relationship between input and target.

ANNs have been applied in several scientific fields and addressed efficiently and effectively a number of hard and complex problems. Some classes of ANN applications are *function approximation*, aiming at learning the functional relationship between the inputs and the desired output, *optimization*, i.e., finding the optimal parameter values in an optimization problem, *data mining*, aiming at discovering hidden patterns in data, *classification*, i.e. prediction of the class of an input vector, *pattern matching*, where the scope is to produce a pattern that is best associated with a given input vector, *pattern completion*, where the scope is to complete missing parts of a given input vector, and *control*, where, given an input vector, an appropriate action is suggested [30].

### 1.2.3 Fuzzy systems

Traditional set theory and binary-valued logic both require two values of parameters, be part of a set or not, and 0 or 1, respectively. Human reasoning, however, includes a measure of uncertainty, and hence is not exact. Fuzzy sets and fuzzy logic allow what is referred to as approximate reasoning. With *fuzzy sets*, an element belongs to a set with a certain degree of certainty. *Fuzzy logic* allows reasoning with these uncertain facts to infer new facts, with a degree of certainty associated with each fact. In a sense, fuzzy sets and fuzzy logic allow the modeling of common sense [30]. The uncertainty in fuzzy systems is referred to as non statistical uncertainty, which should not be confused with statistical uncertainty. Statistical uncertainty is based on the laws of probability, whereas non statistical uncertainty is based on vagueness, imprecision and/or ambiguity. Statistical uncertainty is resolved through observations. For example, when a coin is tossed we are certain what the outcome is, while before tossing the coin, we know that the probability of each outcome is 50%. Nonstatistical uncertainty, or fuzziness, is an inherent property of a system and cannot be altered or resolved by observation. Fuzzy systems have been applied to control systems, gear transmission and braking systems in vehicles, controlling lifts, home appliances, and controlling traffic signals, among others [30, 128].

## 1.3 Review of Cryptography and Cryptanalysis Through Computational Intelligence

Computational Intelligence methods have been successfully applied in numerous scientific fields. Evolutionary Computation (EC) algorithms share a common characteristic, namely that they do not require good mathematical properties, such as continuity or differentiability, for the objective function of the underlying problem. Therefore, they are applicable to hard real–world optimization problems that involve discontinuous objective functions and/or disjoint search spaces [34, 52, 117]. Artificial Neural Networks (ANNs) have also been applied to many scientific fields and problem classes and provided very promising results, due to their parallel and distributed structure and their inherent ability to adapt, learn and generalize. The use of automated techniques in the design and cryptanalysis of cryptosystems is desirable as it minimizes the need for time-consuming human interaction with the search process [17]. However, due to its nature the field of cryptography and cryptanalysis is quite demanding and complex. Thus, the application of an efficient and effective tool such as Computational Intelligence (CI) to the field of cryptology comes naturally. A brief survey of the research relating the two fields follows.

The works of Peleg and Rosenfeld [100] in 1979, Hunter and McKenzie [47] in 1983, Carrol and Martin [13] in 1986 and King and Bahler [53] in 1992, that used relaxation algorithms for breaking simple substitution ciphers, can be considered as predecessors to the application of EC methods in cryptanalysis. In 1993 Spillman et al. [123, 124] and Mathews [77] introduced the use of genetic algorithms for addressing simple substitution, transposition and knapsack ciphers, while later in the same year Forsyth and Safavi-Naini [36] proposed the simulated annealing method for attacking a simple substitution algorithm. In 1995, Jakobsen [49] proposed some simplified hill-climbing techniques for addressing the problems of [36, 124] and in 1996 Vertan and Geangala [129] used genetic algorithms for breaking the Merkle–Hellman cryptosystem. Also, in 1997 Bagnall et al. [4] presented a ciphertext-only attack for a simplified version of an Enigma rotor machine using genetic algorithms.

In 1998 A. Clark proposed in his Ph.D. thesis [17] the tabu search algorithm for cryptanalysis and compared several heuristic techniques, including genetic algorithms, for breaking classical cryptosystems. In his thesis, it was also proved that the knapsack cipher attack of [123] was flawed and, furthermore, Millan, A. Clark and Dawson proposed the construction of Boolean functions with good cryptographic properties utilizing smart hill-climbing techniques and genetic algorithms [88, 89]. Continuing the work of Millan, A. Clark and Dawson, in [18] J. Clark and Jacob presented a two stage optimization for the design of Boolean functions and more recently they proposed new attacking techniques of cryptographic primitives based on fault injection and timing analysis which are effective in breaking a specific kind of identification schemes using simulated annealing [19]. In [12] Burnett et al. designed the S-boxes of MARS, one of the five AES finalists, using hill-climbing and genetic algorithms. Also, J. Clark et al. in [20] provided an improvement for the design of S-boxes using simulated annealing. In 2002 Hernández et al. proposed a new cryptanalytic technique for TEA with reduced number of rounds, which also proved to be useful in distinguishing a block cipher from a random permutation, by applying genetic algorithms [43, 44]. Finally, in 2004 Barbieri et al. [5] described a method for generating good linear block error-correcting codes that uses genetic algorithms, following the idea of genetic approach to code generation of Dontas and Jong [26].

Over the last fifteen years, just a few research studies have appeared to relate ANNs with cryptography and cryptanalysis. In 1991 Rivest wrote a survey article about the relationship between cryptography and machine learning [111], emphasizing on how these fields can contribute ideas to each other. Blum, Furst, Kearns and Lipton in [9] presented how to construct good cryptographic primitives based on problems in learning that are believed to be intractable. Working on the same concept, Pointcheval in [105, 106] used an NP-Complete problem based on ANNs for the design of certain type of secure identification problems but later Knudsen and Meier in [56] demonstrated that this scheme is less secure than what was previously believed. In 1998, Ramzan in his Ph.D. thesis [107] broke the Unix Crypt cryptosystem, a simlified variant of the Enigma cryptosystem, using ANNs. In 2001, an application of cryptology to the field of ANNs was proposed by Chang and Lu [14]. Specifically, they proposed oblivious polynomial evaluation protocols that can operate directly to floating point numbers and gave as example the oblivious learning of an ANN. Also, in [135] a general paradigm for building ANNs for visual cryptography is presented. In [50, 51] Karras and Zorkadis used Feedforward and Hopfield neural networks to improve and strengthen traditional pseudorandom stream generators for the secure management of communication systems. ANNs have also been used for the development of a new key exchange system which is based on a new phenomenon, the synchronization of ANNs [54, 55, 113]. The synchronization of ANNs is a kind of mutual training of ANNs on common inputs. However, in [55] it was shown that this key exchange protocol can be broken in three different ways, using genetic algorithms, genetic attack and probabilistic attack, respectively. Lately, in [91,115,116] some techniques for the improvement of the previously proposed protocol were presented. Finally, the idea of applying ANNs for the construction of S-boxes was presented in [60].

## 1.4 Applying Computational Intelligence in Cryptanalysis

In the following sections our results obtained from the application of CI methods in the cryptanalysis of known cryptosystems are presented. Specifically, in the first section cryptographic problems derived from classical public key cryptosystems are formulated as discrete optimization tasks and EC methods are applied to address them. In the next section, EC methods are considered for the partial cryptanalysis of Feistel ciphers. The effectiveness of ANNs for classical cryptographic problems and problems of elliptic curve cryptography, follow. Lastly, the relationship between a specific class of ANNs, namely the Ridge Polynomial Networks, and theoretical results of cryptography is presented.

### 1.4.1 Cryptanalysis as Discrete Optimization Task

In this section three problems encountered in the field of cryptology are formulated as discrete optimization tasks and two EC algorithms, namely PSO method and DE algorithm, are applied for their cryptanalysis. The reported results suggest that the formulation of the problems as discrete optimization tasks preserves their complexity which makes it difficult for the methods to extract pieces of information [64, 70]. This fact suggests that the main issue when using EC methods in cryptanalysis is the proper definition of the fitness function, i.e., avoiding the deceptive landscapes that lead in results not better than random search, which was also later mentioned in [48]. Thus, the first conclusion derived by these experiments is that, due to the proven complexity of the cryptographic problems, when EC methods are applied to cryptanalysis special attention must be paid to the design of the fitness function so as to include as much information as possible for the target problem. The second conclusion is that EC methods (and CI methods in general) can be used as a quick practical assessment for the efficiency and the effectiveness of proposed cryptographic systems. Specifically, since strong cryptosystems must not reveal any patterns of the encrypted messages or their inner structure (as this could lead to their cryptanalysis), CI methods can be used as a first measure for the evaluation of new cryptographic schemes before more formal methods (which may be complex to apply) are employed for their analysis.

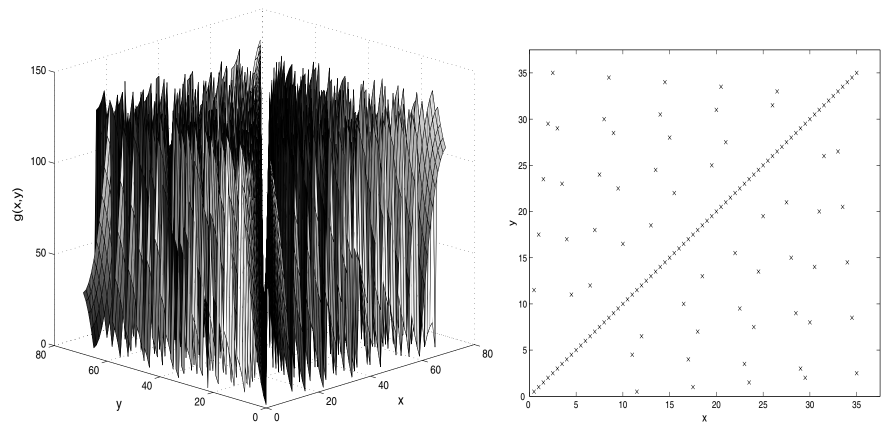

**Fig. 1.2.** (a) Plot of function $g(x,y) = x^2 - y^2 (\text{mod } N)$, for $N = 143$ and (b) contour plot of function $g(x,y) = x^2 - y^2 (\text{mod } N)$, for $N = 143$ at value $g = 0$

## Problem Formulation

All three problems considered below are derived from the factorization problem described in Sect. 1.1.2. The first problem is defined as follows: given a composite integer $N$, find pairs of $x$, $y \in \mathbb{Z}_N^*$, such that $x^2 \equiv y^2 (\text{mod } N)$, with $x \not\equiv \pm y (\text{mod } N)$. This problem is equivalent to finding non-trivial factors of $N$, as $N$ divides $x^2 - y^2 = (x-y)(x+y)$, but $N$ does not divide either $x - y$ or $x + y$. Thus, the $\gcd(x-y, N)$ is a non-trivial factor of $N$ [85].

We formulate the problem as a discrete optimization task by defining the minimization function $f : \{1, 2, \ldots, N–1\} \times \{1, 2, \ldots, N–1\} \mapsto \{0, 1, \ldots, N–1\}$, with

$$f(x,y) = x^2 - y^2 \pmod N, \tag{1.21}$$

subject to the constraints $x \not\equiv \pm y (\text{mod } N)$. The constraint $x \equiv -y (\text{mod } N)$ can be incorporated in the problem by changing the function domain. In this case, the problem reduces to minimizing the function $g : \{2, 3, \ldots, (N-1)/2\} \times \{2, 3, \ldots, (N-1)/2\} \mapsto \{0, 1, \ldots, N–1\}$, with

$$g(x,y) = x^2 - y^2 \pmod N, \tag{1.22}$$

subject to the constraint $x \not\equiv y (\text{mod } N)$. This is a 2-dimensional minimization problem and the global minimum of the function $g$ is zero. For simplicity, we will call the minimization problem of function $g$, i.e., finding a global minimizer $(x^*, y^*)$ of the function $g$, subject to the constraint $x \not\equiv y (\text{mod } N)$, as **Problem 1**. An illustration of the function $g(x,y)$ for $N = 11 \times 13 = 143$ is depicted in Fig. 1.2 (a), and the contour plot of function $g$ at the global minimum $g(x,y) = 0$ is shown in Fig. 1.2 (b).

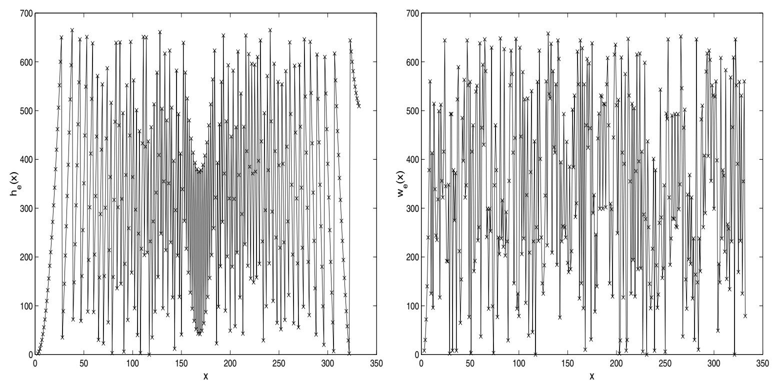

**Fig. 1.3.** (a) Plot of function $h_e(x) = (x-1)(x-2)(\bmod N)$, for $N = 667$ and (b) plot of function $w_e(x) = (x+1)(x-1)(x-2)(\bmod N)$, for $N = 667$

We also consider the following minimization problems. Let us define the minimization function $h : \{1, 2, \ldots, N–1\} \mapsto \{0, 1, \ldots, N–1\}$, with

$$h(x) = (x-a)(x-b) \pmod N, \tag{1.23}$$

where $a, b$ non–zero integers and $x \not\equiv a(\bmod N)$, $x \not\equiv b(\bmod N)$. A test case of this problem is the function

$$h_e(x) = (x-1)(x-2) \pmod N, \tag{1.24}$$

where $x \not\equiv 1(\bmod N)$ and $x \not\equiv 2(\bmod N)$. This is 1-dimensional minimization problem with global minimum zero. We will refer to the minimization of function $h_e(x)$, subject to the constraints $x \not\equiv 1(\bmod N)$ and $x \not\equiv 2(\bmod N)$, as **Problem 2**. Figure 1.3 (a) depicts the function $h_e(x)$ for the small value of $N = 23 \times 29 = 667$.

In a more general setting, we can consider the minimization of the function

$$w(x) = (x-a)(x-b)\cdots(x-m) \pmod N, \tag{1.25}$$

where $x \in \{0, 1, \ldots, N–1\}$ and $x \not\equiv \{a, b, \ldots, m\}(\bmod N)$. We study the case function

$$w_e(x) = (x+1)(x-1)(x-2) \pmod N, \tag{1.26}$$

with $x \not\equiv \{-1, 1, 2\}(\bmod N)$. We will refer to the 1-dimensional minimization of function $w_e(x)$, subject to the constraints with $x \not\equiv \{-1, 1, 2\}(\bmod N)$, as **Problem 3.** In Fig. 1.3 (b) an illustration of the function $w_e(x)$ for the small value $N = 23 \times 29$ is shown.

## Experimental Setup and Results

Both Particle Swarm Optimization (PSO) [21] and Differential Evolution (DE) [126] methods are applied on the problems formulated in the previous section (Problems 1,2,3) and compared with the simple random search technique. The global and local PSO variants of both the inertia weight and the constriction factor versions, as well as the DE variants with the mutation operators of (1.5) and (1.8), are used. The typical parameter values for the PSO variants are used (see Sect. 1.2), and the local variant of the PSO is tested for neighborhood size equal to 1. For the PSO, preliminary experiments on the specific problems indicated that the value of maximum velocity $V_{\max}$ of the particles affects its performance significantly. The most promising results were produced using the values $V_{\max} = ((N-7)/10, (N-7)/10)$ for Problem 1, and the value $V_{\max} = (N-4)/5$ for Problems 2 and 3, and therefore they are adopted in all the experiments. The parameters of the DE algorithm are set at the values $F = 0.5$ and $CR = 0.5$. In all cases, the populations are constrained to lie within the feasible region of the corresponding problem.

For the minimization of function $g$ (Problem 1), the performance of the methods is investigated for several values of $N$, in the range $N = 199 \times 211 = 41\ 989$ up to $N = 691 \times 701 = 484\ 391$. For each value of $N$ considered, 100 independent runs are performed. The corresponding results are shown in Table 1.1. In this table, PSOGW corresponds to the global variant of PSO method with inertia weight; PSOGC is the global variant of PSO with constriction factor; PSOLW is PSO's local variant with inertia weight; PSOLC is PSO's local variant with constriction factor, DE1 corresponds to the DE algorithm with the mutation operator of (1.5) and DE2 to the DE algorithm with the mutation operator of (1.8). Random search results are denoted as RS. A run is considered to be successful if the algorithm identifies a global minimizer within a prespecified number of function evaluations. The function evaluations threshold is taken equal to the cardinal of integers in the domain of the target function. The success rates of each algorithm, i.e. the fraction of times it achieved a global minimizer within the prespecified threshold, the minimum number, the median, the mean value and the standard deviation of function evaluations (F.E.) needed for success, are also reported in the table.

The experimental results of Table 1.1 indicate that the variants of the PSO method outperform in success rates the variants of the DE method over these problem instances (i.e., different values of the parameter $N$) and with the parameter setup used. Moreover, the performance of the DE method decreases as the value of $N$ increases while PSO appears to be more stable with respect to this parameter. However, in contrast to the known behavior of EC methods, the best success rates achieved are relatively low (around 50%) and the random search technique (RS) outperforms both EC methods and their variants. This fact suggests that the almost random behavior of the specific kind of problem makes it quite difficult for the methods to extract knowledge about their dynamics. In the cases where EC methods located a global minimizer they

**Table 1.1.** Results for the minimization of function $g$ (see (1.22))

| $N$ | Method | Suc.Rate | mean F.E. | St.D. F.E. | median F.E. | min F.E. |
|---|---|---|---|---|---|---|
| | PSOGW | 56% | 8844.643 | 5992.515 | 8325.000 | 660 |
| | PSOGC | 48% | 7149.375 | 5272.590 | 5355.000 | 330 |
| | PSOLW | 51% | 8329.412 | 6223.142 | 7050.000 | 270 |
| $N = 199 \times 211$ | PSOLC | 51% | 7160.588 | 6001.276 | 5940.000 | 420 |
| | DE1 | 4% | 517.500 | 115.866 | 465.000 | 450 |
| | DE2 | 9% | 5476.667 | 6455.651 | 1830.000 | 60 |
| | RS | 66% | 9104.015 | 5862.358 | 8700.500 | 22 |
| | PSOGW | 41% | 16210.244 | 11193.375 | 15090.000 | 120 |
| | PSOGC | 45% | 16818.667 | 12664.632 | 13800.000 | 630 |
| | PSOLW | 58% | 18455.690 | 12870.897 | 14520.000 | 270 |
| $N = 293 \times 307$ | PSOLC | 50% | 16374.000 | 13597.782 | 13365.000 | 120 |
| | DE1 | 7% | 1598.571 | 1115.488 | 1470.000 | 120 |
| | DE2 | 19% | 17815.263 | 12484.580 | 16290.000 | 2730 |
| | RS | 64% | 21548.531 | 13926.751 | 20852.500 | 57 |
| | PSOGW | 53% | 31965.849 | 24423.975 | 27570.000 | 780 |
| | PSOGC | 45% | 32532.667 | 22652.983 | 33210.000 | 1740 |
| | PSOLW | 55% | 31472.182 | 23394.791 | 22620.000 | 720 |
| $N = 397 \times 401$ | PSOLC | 54% | 38156.111 | 22925.970 | 37665.000 | 750 |
| | DE1 | 1% | 1680.000 | 0.000 | 1680.000 | 1680 |
| | DE2 | 12% | 27722.500 | 17498.736 | 28620.000 | 180 |
| | RS | 60% | 27302.567 | 21307.031 | 23607.500 | 145 |
| | PSOGW | 56% | 49893.750 | 37515.327 | 44640.000 | 930 |
| | PSOGC | 55% | 49975.636 | 36727.380 | 41760.000 | 300 |
| | PSOLW | 55% | 49207.091 | 34053.904 | 50430.000 | 2010 |
| $N = 499 \times 503$ | PSOLC | 46% | 48443.478 | 34677.039 | 43470.000 | 1920 |
| | DE1 | 1% | 2480.000 | 0.000 | 2480.000 | 2480 |
| | DE2 | 8% | 67245.000 | 35114.316 | 64770.000 | 14730 |
| | RS | 61% | 54139.443 | 38642.970 | 48743.000 | 140 |
| | PSOGW | 52% | 72175.000 | 48653.823 | 71550.000 | 600 |
| | PSOGC | 51% | 81476.471 | 53666.543 | 75100.000 | 5000 |
| | PSOLW | 49% | 78651.020 | 48197.105 | 67400.000 | 11200 |
| $N = 599 \times 601$ | PSOLC | 52% | 69542.308 | 48837.949 | 53050.000 | 2500 |
| | DE1 | 2% | 4700.000 | 4808.326 | 4700.000 | 1300 |
| | DE2 | 5% | 8620.000 | 8078.180 | 9300.000 | 800 |
| | RS | 64% | 86123.656 | 47504.284 | 89392.500 | 904 |
| | PSOGW | 46% | 207443.478 | 163585.340 | 214800.000 | 800 |
| | PSOGC | 46% | 175426.086 | 138118.794 | 149200.000 | 800 |
| | PSOLW | 60% | 196993.334 | 146204.518 | 144500.000 | 9200 |
| $N = 691 \times 701$ | PSOLC | 52% | 209307.692 | 163833.606 | 200100.000 | 1800 |
| | DE1 | 2% | 23800.000 | 25000.000 | 23800.000 | 21000 |
| | DE2 | 10% | 71000.000 | 95357.642 | 15200.000 | 1600 |
| | RS | 60% | 185932.334 | 126355.926 | 154999.000 | 2828 |

required a quite small number of function evaluations with respect to the cardinal of the domain of the function. Finally, it is important to note that in the experiments where the EC methods failed in obtaining a global minimizer, they located a local minimizer with value close to the global minimum.

Similar results are obtained for the minimization of the functions $h_e$ (Problem 2) and $w_e$ (Problem 3), and for $N = 103 \times 107$ they are reported in Table 1.2. In the case of Problem 3, the success rates of PSO method and its variants are high (around 80%) while the performance of DE variants remains low. However, random search again outperforms the EC methods in terms of success rates.

**Table 1.2.** Results for functions $h_e$ (see (1.24)) and $w_e$ (see (1.26)), for $N = 103 \times 107$

| Function | Method | Suc.Rate | mean F.E. | St.D. F.E. | median F.E. | min F.E. |
|---|---|---|---|---|---|---|
| | PSOGW | 51% | 2013.333 | 1483.535 | 1500.000 | 100 |
| | PSOGC | 57% | 1974.035 | 1609.228 | 1420.000 | 60 |
| | PSOLW | 59% | 1677.288 | 1254.688 | 1420.000 | 60 |
| $h_e$ | PSOLC | 58% | 2385.862 | 1676.898 | 2040.000 | 120 |
| | DE1 | 1% | 100.000 | 0.000 | 100.000 | 100 |
| | DE2 | 1% | 80.000 | 0.000 | 80.000 | 80 |
| | RS | 65% | 2099.646 | 1448.007 | 2056.000 | 6 |
| | PSOGW | 79% | 1382.785 | 1265.927 | 820.000 | 40 |
| | PSOGC | 84% | 1402.857 | 1442.194 | 930.000 | 40 |
| | PSOLW | 80% | 1757.750 | 1544.267 | 1110.000 | 40 |
| $w_e$ | PSOLC | 85% | 1416.000 | 1329.034 | 880.000 | 40 |
| | DE1 | 1% | 60.000 | 0.000 | 60.000 | 60 |
| | DE2 | 1% | 80.000 | 0.000 | 80.000 | 80 |
| | RS | 96% | 1507.969 | 1328.913 | 1104.000 | 7 |

### 1.4.2 Cryptanalysis of Feistel Ciphers through Evolutionary Computation Methods

In this section two different instances of a problem introduced by the Differential Cryptanalysis of a Feistel cryptosystem are considered and formulated as optimization tasks. Specifically, the problem of finding some missing bits of the key that is used in a simple Feistel cipher, namely the Data Encryption Standard with four and six rounds, respectively is studied [65, 66]. The two instances are complementary, since every problem of missing bits of keys in Differential Cryptanalysis of Feistel ciphers can be categorized into one of the two cases.

The performance of PSO and the DE on this problem is studied. Experimental results for DES reduced to four rounds show that the optimization methods considered, located the solution efficiently, as they required a smaller number of function evaluations compared to the brute force approach. For DES reduced to six rounds the effectiveness of the proposed algorithms depends on the construction of the objective function.

## Problem Formulation

*DES reduced to four rounds*

For DES reduced to four rounds Differential Cryptanalysis (DC) uses a one–round characteristic occurring with probability 1, recovering at the first step of the cryptanalysis 42 bits of the subkey of the last round. Considering the case where the subkeys are calculated using the DES key scheduling algorithm, the 42 bits given by DC are actual key bits of the key and there are 14 key bits still missing for the completion of the key. The brute force attack (i.e. search among all 14-bit keys) requires testing $2^{14}$ trials. The right key should satisfy the known plaintext XOR value for all the pairs that are used by DC. An alternative approach is to use a second characteristic that corresponds to the missing bits and attempt a more careful counting on the key bits of the last two rounds, which is however more complicated.

Instead of using the aforementioned approaches to find the missing key bits, we formulate the problem of computing the missing bits as an integer optimization problem [71]. Since the right key should satisfy the known plaintext XOR value for all the pairs that are used by DC, these ciphertexts can be used for the evaluation of possible solutions provided by optimization methods. Thus, let $\mathbf{X}$ be a 14–dimensional vector, where each of its components corresponds to one of the 14 unknown key bits. Such a vector represents a possible solution of the optimization problem. Also, let $np$ be the number of ciphertext pairs used by DC to obtain the right 42 key bits. Then we can construct the 56 bits of the key, using the 42 bits which are recovered by DC and the 14 components of $\mathbf{X}$ in the proper order. With the resulting key, decrypt the $np$ ciphertext pairs and count the number of decrypted pairs that satisfy the known plaintext XOR value, denoted as $cnp_{\mathbf{X}}$. Thus, the objective function $f$ is the difference between the desired output $np$ and the actual output $cnp_{\mathbf{X}}$, i.e. $f(\mathbf{X}) = np - cnp_{\mathbf{X}}$. The global minimum of the function $f$ is zero and the global minimizer is with high probability the actual key. A first study of this approach is given in [64].

*DES reduced to six rounds*

The cryptanalysis of DES reduced to six rounds is, as expected, more complicated than that of the four round version, since the best characteristic that can be used has probability less than 1. In particular, DC uses two characteristics of probability $p_{sr} = 1/16$ to provide 42 bits of the right key. Again, there are 14 bits of the key missing. However, in this case the right key may not be suggested by all ciphertext pairs. This happens because not all the corresponding plaintexts pairs are *right pairs*. A pair is called *right pair with respect to an $r$-round characteristic $\Omega = (\Omega_P, \Omega_\Lambda, \Omega_C)$ and an independent key $K$*, if it holds that $P' = \Omega_P$, where $P'$ is the pair's XOR value, and for the first $r$ rounds of the encryption of the pair using the independent key $K$ the input and output XORs of the $i$th round are equal to $\lambda_I^i$ and $\lambda_O^i$, respectively [6].

The probability that a pair with plaintext XOR equal to $\Omega_P$ of the characteristic is a right pair using a fixed key is approximately equal to the probability of the characteristic. A pair which is not a right pair is called *wrong pair* and it does not necessarily suggest the right key as a possible value. The study of right and wrong pairs has shown that the right key appears with the probability of the characteristic from the right pairs and some other random occurrences from wrong pairs. In conclusion, if all the pairs of DC (right and wrong) are used in the predefined objective function $f$, the function's minimum value will change depending on the specific pairs used. On the other hand, if the right pairs are filtered and are solely used in the objective function $f$, the function's global minimum will be constant with value equal to 0, as in the case of missing bits of DES reduced to four rounds. As the filtering of the right pairs is not always possible and easy, we study the behavior of the proposed approach using the objective function $f$ with all the pairs of DC.

### Experimental Setup and Results

Both the PSO and DE methods were applied considering each component of the possible solution as a real number in the range $[0, 1]$ and all populations were constrained to lie within the feasible region of the problem. For the evaluation of the suggested solutions, the technique of rounding off the real values of the solution to the nearest integer [71, 108] was applied. For the PSO method we have considered both the global and local variants, and for the DE algorithm all five variants described in Sect. 1.2. A maximum value for the velocity, $V_{\max} = 0.5$, of the PSO method was set in order to avoid the swarm's explosion, i.e. avoid velocities from assuming large values that lead to fluctuation of the particles over the search space, and thus destroy the dynamic of the method. The parameters of PSO were set at the default values, i.e. $\chi = 0.729$ and $c_1 = c_2 = 2.05$, found in the literature [21], and the parameters of DE were set at equal values $CR = F = 0.5$.

The proposed approach was tested for several different initial keys and number of pairs, *np*. For each setting, the size of each population was equal to 100 and the performance of the methods was investigated on 100 independent runs. A run is considered to be successful if the algorithm identifies the global minimizer within a prespecified number of function evaluations. The function evaluations threshold for both problems was taken equal to $2^{14}$. For the missing bits of the key of DES reduced to four rounds, the results for six different keys, $k_i$, $i = 1, 2, \ldots, 6$, and for test pairs, *np*, equal to 20 and 50 are reported in Tables 1.3, 1.4, respectively. In the tables, PSOCG denotes the global variant of the constriction factor version of PSO, PSOCL1 denotes the PSO's local variant with neighborhood size equal to 1, PSOCL2 corresponds to PSO's local variant with neighborhood size equal to 2 (see Sect. 1.2), and DE1, DE2, DE3, DE4, DE5 denote the five DE variants of (1.5),(1.8),(1.9),(1.10),(1.11), respectively. Each table reports the success rate (Suc.Rate) of each algorithm, that is the proportion of the times it achieved the global minimizer within the prespecified threshold, and the mean value of function evaluations (Mean F.E.) over the successful experiments.

The results for the problem of recovering the missing (after the application of DC) bits of DES reduced to four rounds suggest that the proposed approach is able to locate the global minimizer i.e. the 14 missing bits of the key, with relatively low computational cost compared to the brute force attack. The success rates of all versions of the two methods are high. For $np$ equal to 20 (Table 1.3) success rates range from 93% to 100%, with an average of 99.3%. For $np$ equal to 50 (Table 1.4) the success rates lie in the region from 90% to 100%, with mean 99.4%. The improvement of the success rate as the number of ciphertext pairs, $np$, increases is expected as the larger number of ciphertext pairs used for the evaluation of the possible solutions reduces the possibility of a wrong 14-tuple being suggested as the right one. The mean number of function evaluations required to locate the global minimizer (over all variants) is 1309 in the case of $np = 20$ and 982 in the case of $np = 50$. This implies that, as more ciphertext pairs are incorporated in the objective function, not only the evaluation becomes more accurate, but also the global minimizer becomes easier to locate. However, the number of ciphertext pairs used by the proposed approach should not exceed the number of the ciphertext pairs that are used by DC for the initial problem, as this would increase the total cost of the cryptanalysis in terms of encryptions and decryptions.

With respect to the different variants of the PSO method, the local variant with neighborhood size 2 (PSOLC2) accomplished success rates close to 100%, in all instances of the first problem, with an average of 1489 function evaluations. The global variant of PSO (PSOGC) achieved success rates from 93% to 100% in different instances of the problem, but with an average of 898 function evaluations. This means that, although the global variant of PSO exhibits overall lower success rates, in the cases where both local and global variants of PSO are able to locate the minimizer, the global variant requires less function evaluations than the local variant.

The DE variants exhibited a stable and similar to each other behavior, with mean success rates 100% in all cases. One minor exception was DE4 that achieved mean success rates of 99% on two instances of the problem. DE1 required the lowest mean number of function evaluations (576) among all the considered methods and their variants.

For the missing bits of the key of DES reduced to six rounds, where both right and wrong pairs are used in construction of the objective function, the

**Table 1.3.** Results for DES reduced to four rounds for six different keys using $np = 20$ test pairs

| key | Method | Suc.Rate | Mean F.E. |
|---|---|---|---|
| | PSOGC | 99% | 742.42 |
| | PSOLC1 | 100% | 1773.00 |
| | PSOLC2 | 100% | 1255.00 |
| | DE1 | 100% | 614.00 |
| $k_1$ | DE2 | 100% | 1406.00 |
| | DE3 | 100% | 780.00 |
| | DE4 | 100% | 588.00 |
| | DE5 | 100% | 1425.00 |
| | PSOGC | 99% | 911.11 |
| | PSOLC1 | 100% | 2665.00 |
| | PSOLC2 | 100% | 1650.00 |
| | DE1 | 100% | 603.00 |
| $k_2$ | DE2 | 100% | 1518.00 |
| | DE3 | 100% | 879.00 |
| | DE4 | 100% | 615.00 |
| | DE5 | 100% | 1649.00 |
| | PSOGC | 94% | 1117.02 |
| | PSOLC1 | 99% | 2447.48 |
| | PSOLC2 | 100% | 1688.00 |
| | DE1 | 99% | 693.94 |
| $k_3$ | DE2 | 100% | 1497.00 |
| | DE3 | 100% | 805.00 |
| | DE4 | 100% | 690.00 |
| | DE5 | 100% | 1427.00 |
| | PSOGC | 96% | 876.04 |
| | PSOLC1 | 100% | 2089.00 |
| | PSOLC2 | 100% | 1418.00 |
| | DE1 | 99% | 701.01 |
| $k_4$ | DE2 | 100% | 1378.00 |
| | DE3 | 100% | 843.00 |
| | DE4 | 100% | 568.00 |
| | DE5 | 100% | 1362.00 |
| | PSOGC | 97% | 900.00 |
| | PSOLC1 | 99% | 1979.80 |
| | PSOLC2 | 100% | 1496.00 |
| | DE1 | 100% | 662.00 |
| $k_5$ | DE2 | 100% | 1493.00 |
| | DE3 | 100% | 848.00 |
| | DE4 | 100% | 662.00 |
| | DE5 | 100% | 1542.00 |
| | PSOGC | 93% | 1457.00 |
| | PSOLC1 | 95% | 4475.79 |
| | PSOLC2 | 99% | 2913.13 |
| | DE1 | 100% | 651.00 |
| $k_6$ | DE2 | 100% | 1717.00 |
| | DE3 | 100% | 1063.00 |
| | DE4 | 99% | 725.25 |
| | DE5 | 100% | 1583.00 |

**Table 1.4.** Results for DES reduced to four rounds for six different keys using $np = 50$ test pairs

| key | Method | Suc.Rate | Mean F.E. |
|---|---|---|---|
| | PSOGC | 99% | 860.61 |
| | PSOLC1 | 100% | 1698.00 |
| | PSOLC2 | 100% | 1141.00 |
| | DE1 | 100% | 485.00 |
| $k_1$ | DE2 | 100% | 1215.00 |
| | DE3 | 100% | 785.00 |
| | DE4 | 100% | 553.00 |
| | DE5 | 100% | 1382.00 |
| | PSOGC | 94% | 741.49 |
| | PSOLC1 | 100% | 1367.00 |
| | PSOLC2 | 100% | 1100.00 |
| | DE1 | 99% | 490.91 |
| $k_2$ | DE2 | 100% | 1081.00 |
| | DE3 | 100% | 669.00 |
| | DE4 | 100% | 521.00 |
| | DE5 | 100% | 1128.00 |
| | PSOGC | 99% | 631.31 |
| | PSOLC1 | 100% | 1217.00 |
| | PSOLC2 | 100% | 1035.00 |
| | DE1 | 100% | 385.00 |
| $k_3$ | DE2 | 100% | 1006.00 |
| | DE3 | 100% | 546.00 |
| | DE4 | 100% | 409.00 |
| | DE5 | 100% | 1016.00 |
| | PSOGC | 90% | 947.78 |
| | PSOLC1 | 98% | 2292.88 |
| | PSOLC2 | 100% | 1588.00 |
| | DE1 | 98% | 666.33 |
| $k_4$ | DE2 | 100% | 1342.00 |
| | DE3 | 100% | 838.00 |
| | DE4 | 99% | 649.50 |
| | DE5 | 100% | 1294.00 |
| | PSOGC | 100% | 707.00 |
| | PSOLC1 | 100% | 1763.00 |
| | PSOLC2 | 100% | 1193.00 |
| | DE1 | 100% | 445.00 |
| $k_5$ | DE2 | 100% | 1127.00 |
| | DE3 | 100% | 684.00 |
| | DE4 | 100% | 465.00 |
| | DE5 | 100% | 1131.00 |
| | PSOGC | 96% | 880.21 |
| | PSOLC1 | 100% | 2009.00 |
| | PSOLC2 | 100% | 1390.00 |
| | DE1 | 100% | 507.00 |
| $k_6$ | DE2 | 100% | 1250.00 |
| | DE3 | 100% | 692.00 |
| | DE4 | 100% | 563.00 |
| | DE5 | 100% | 1230.00 |

**Table 1.5.** Results for DES reduced to six rounds for six different keys using $np = 200$ test pairs

| key | Method | Suc.Rate | Mean F.E. |
|---|---|---|---|
| $k_1$ | PSOGC | 26% | 7038.46 |
| | PSOLC1 | 9% | 2188.89 |
| | PSOLC2 | 8% | 3862.50 |
| | DE1 | 36% | 5191.67 |
| | DE2 | 52% | 5515.39 |
| | DE3 | 41% | 5807.32 |
| | DE4 | 51% | 6364.71 |
| | DE5 | 59% | 6855.93 |
| $k_2$ | PSOGC | 24% | 5037.50 |
| | PSOLC1 | 3% | 1500.00 |
| | PSOLC2 | 7% | 2357.14 |
| | DE1 | 34% | 6535.29 |
| | DE2 | 58% | 6968.97 |
| | DE3 | 40% | 5945.00 |
| | DE4 | 39% | 6897.44 |
| | DE5 | 61% | 6932.79 |
| $k_3$ | PSOGC | 41% | 4902.44 |
| | PSOLC1 | 6% | 4533.33 |
| | PSOLC2 | 5% | 7340.00 |
| | DE1 | 48% | 5070.83 |
| | DE2 | 61% | 6967.21 |
| | DE3 | 53% | 6698.11 |
| | DE4 | 48% | 5889.58 |
| | DE5 | 56% | 7926.79 |
| $k_4$ | PSOGC | 47% | 4912.77 |
| | PSOLC1 | 13% | 4407.69 |
| | PSOLC2 | 23% | 4134.78 |
| | DE1 | 57% | 6491.23 |
| | DE2 | 76% | 7594.74 |
| | DE3 | 66% | 6418.18 |
| | DE4 | 72% | 5741.67 |
| | DE5 | 76% | 7001.32 |
| $k_5$ | PSOGC | 36% | 5575.00 |
| | PSOLC1 | 4% | 1950.00 |
| | PSOLC2 | 5% | 4700.00 |
| | DE1 | 51% | 5688.24 |
| | DE2 | 62% | 7803.23 |
| | DE3 | 57% | 5229.83 |
| | DE4 | 53% | 5377.36 |
| | DE5 | 64% | 6387.50 |
| $k_6$ | PSOGC | 37% | 5624.32 |
| | PSOLC1 | 5% | 2920.00 |
| | PSOLC2 | 9% | 3377.78 |
| | DE1 | 49% | 5681.63 |
| | DE2 | 63% | 7380.95 |
| | DE3 | 50% | 7048.00 |
| | DE4 | 51% | 5621.57 |
| | DE5 | 64% | 7679.69 |

results for the same six different keys tested for DES reduced to four rounds, and for test pairs, $np$, equal to 200, are reported in Table 1.5.

From Tables 1.3,1.4,1.5, we observe that there is a considerable difference between the success rates for the case of four rounds and the case of six rounds. This can be attributed to the fact that in the former case we work with a characteristic that occurs with probability 1 while in the latter case we work with a characteristic with smaller probability (1/16). This means that in the set of 200 ciphertext pairs used by the objective function, approximately 12 pairs are right and suggest the right tuple, while the remaining 188 pairs suggest tuples at random, thus decreasing, the possibility of suggestion of the right tuple. Consequently, since the objective function becomes more effective when more right pairs are available or equivalently, when the probability of the utilized characteristic is large, it is expected that in the four round case the performance of the methods should be better than in the six round case. Although, the wrong pairs used in the objective function of DES for six rounds are misleading for the evaluation of the right tuple of missing bits, the global variant of PSO (PSOGC) and all DE variants (DE1–DE5) were able to locate the missing bits on an average of 35% of independent runs for PSOGC and 55% for the DE variants over all six different keys tested (Table 1.5). The function evaluations required for the location of the right 14–tuple of missing bits in this case are on average 5600 for all methods.

Finally, an interesting observation from the results of the proposed approach is that in the case of DES reduced to four rounds all methods in independent runs were able to locate four different 14–tuples satisfying the condition criterion of the objective function. These four solutions of the problem differed in two fixed positions, the 10th and the 36th, of the DES key. In the case of DES reduced to six rounds just one solution, the right one, was located by all methods.

The results indicate that the proposed methodology is efficient in handling this type of problems, since on DES reduced to four rounds it managed to address the problem at hand using an average of 576 function evaluations in contrast with the brute force approach that requires $2^{14} = 16384$ evaluations. Furthermore, the results of DES reduced to six rounds suggest that the effectiveness of the proposed approach depends mainly on the construction of the objective function. This approach is also applicable to all Feistel cryptosystems that are amenable to differential cryptanalysis, thus motivating its use for other Feistel cryptosystems. Finally, as a future direction, we are interested in studying the effectiveness of the proposed approach not just for missing bits of the key produced by Differential Cryptanalysis but also for all the bits of the key of Feistel ciphers.

### 1.4.3 Utilizing Artificial Neural Networks to Address Cryptographic Problems

In this section we consider the Artificial Neural Networks approach and study its performance on some cryptographic problems [69]. Specifically, we study the approximation of the *Discrete Logarithm Problem* (DLP) and the *Diffie Hellman key–exchange protocol Problem* (DHP) over the finite field $\mathbb{Z}_p$, where $p$ is prime, and the *factorization problem related to the RSA cryptosystem* [112] (all three problems are presented in Sect. 1.1.2).

#### Experimental Setup and Results

*Training algorithms:* In this study the ANN training algorithms considered were the Standard Back Propagation (BP) [114], the Back Propagation with Variable Stepsize (BPVS) [75], the Resilient Back Propagation (RPROP) [110], the On-Line Adaptive Back Propagation (OABP) [73] and the Scaled Conjugate Gradient (SCG) method [92]. All methods were extensively tested with a wide range of parameters. In most of the testing cases, the training methods did not exhibit significantly different performance, except from BP, which encountered difficulties in training most of the times.

*Network architecture:* Since the definition of an "optimal" network architecture for any particular problem is quite difficult and remains an open problem, we tested a variety of topologies with different numbers of hidden layers and with various numbers of neurons at each layer. The results reported are the best results obtained for each problem. The architecture used is described with a series of integers denoting the number of neurons at each layer.

*Data normalization:* To make the adaptation of the network easier, the data are transformed through the normalization procedure, that takes place before training. Assuming that the data presented to the network are in $\mathbb{Z}_p$, where $p$ is a prime number, the space $S = [-1, 1]$, is split in $p$ sub-spaces. Thus, numbers in the data set are transformed to analogous ones in the space $S$. At the same time, the network output is transformed to a number within $\mathbb{Z}_p$ using the inverse operation.

*Network evaluation:* For the evaluation of the network performance we first measured the percentage of the training data, for which the network was able to compute the exact target value. This measure is denoted by $\mu_0$. However, as network output was restricted within the range $[-1, 1]$, very small differences in output, rendered the network unable to compute the exact target but rather to be very close to it. This fact resulted in the insufficiency of the $\mu_0$ measure as a performance indicator. Thus we employed the $\mu_{\pm v}$ measure. This measure represents the percentage of the data for which the difference between desired and actual output does not exceed $\pm v$ of the real target.

We note that the "near" measure, $\mu_{\pm v}$, has different meaning for the DLP and the DHMP. The "near" $\mu_{\pm v}$ measure is very important in the case of the DLP. If the size of the $(\pm v)$ interval is $O(\log(p))$ then the "near" measure can replace the "complete" $\mu_0$ one. In general, for some small values of $v$ the "near" measure is acceptable since the discrete logarithm computation can be verified i.e. computation of exponents over finite fields [104]. However, the verification of the Diffie–Hellman Mapping is an open problem. Sets of possible values for the Diffie–Hellman Mapping can be used to compute sets of possible values for the Diffie–Hellman key. The values of the Diffie–Hellman key can be tested in practice; they are symmetric keys of communication between the two users. The percentage of success for the "near" measure for DHMP can be compared with the corresponding percentage for the DLP. The results of the comparison can be related to the conjecture that the two problems are computationally equivalent.

In [84], both DLP and DHMP for several small prime numbers $p$ have been tested. The input patterns were different values of the input variable of the discrete logarithm function and the Diffie–Hellman Mapping respectively, and the target patterns were the values of the corresponding function, for fixed chosen values of generators $g$ and primes $p$. The ANNs in this case succeeded in training and generalizing, reaching up to 100%. Next, larger primes were tested rendering the task of training networks harder. Having so many numbers normalized in the range $[-1, 1]$ posed problems for the adaptation process. Thus, small changes in the network output caused complete failure, requiring the use of larger architectures, i.e., more nodes and layers. In cases with very large primes, the network performance on training was very poor. Some indicative results on training are reported in Table 1.6.

**Table 1.6.** Results for networks trained on the DLP and DHMP

| $p$ | Topology | Epochs | $\mu_0$ | $\mu_{\pm 2}$ | $\mu_{\pm 5}$ | $\mu_{\pm 10}$ | Problem |
|---|---|---|---|---|---|---|---|
| 83 | $1-5-5-1$ | 20000 | 20% | 30% | 48% | 70% | DLP |
| | $1-5-5-1$ | 20000 | 20% | 35% | 51% | 70% | DHMP |
| 97 | $1-5-5-1$ | 25000 | 20% | 30% | 48% | 70% | DLP |
| | $1-5-5-1$ | 20000 | 20% | 35% | 51% | 70% | DHMP |

The DLP is also studied in a different setting. More specifically, we have studied the case where, for several values of the prime $p$ and the primitive root $g$, the value of $h = g^u (\mathrm{mod}\ p)$, remains fixed. The input patterns consisted of pairs of primes $p$ and the corresponding primitive roots $g$ and the target patterns were the corresponding values of $u$, such that $\log_g h \equiv u (\mathrm{mod}\ p)$, for a chosen fixed value $h$. We have tested the DLP in this setting for $p$ assuming values between 101 and 2003, with several network topologies and training methods. In this case, there was a differentiation among the results obtained by different methods. For small values of $p$, i.e. from 101 to 199, the best results on the approximation of $u$, were obtained by the AOBP method. For larger values of $p$, the best results were given by the SCG method. Results on this new setting are reported in Table 1.7. All these results refer to training the ANNs on the approximation of the value of discrete logarithm $u$. Comparing the results exhibited in Tables 1.6 and 1.7, it seems that for the DLP problem, the approximation capability of the FNNs is better in the new setting.

**Table 1.7.** Results for networks trained on the second setting of the DLP

| Range of $p$ | Topology | Epochs | $\mu_0$ | $\mu_{\pm 15}$ | $\mu_{\pm 20}$ | $\mu_{\pm 30}$ | $\mu_{\pm 40}$ |
|---|---|---|---|---|---|---|---|
| $101 - 199$ | $2 - 15 - 1$ | 600000 | 100% | 100% | 100% | 100% | 100% |
| $503 - 1009$ | $2 - 25 - 1$ | 600000 | 82% | 93% | 96% | 96% | 98% |
| $1009 - 2003$ | $2 - 30 - 1$ | 600000 | 17% | 40% | 46.7% | 51.8% | 54.1% |
| $1009 - 2003$ | $2 - 3 - 3 - 3 - 1$ | 20000 | 7.5% | 34.3% | 44.8% | 64.2% | 71.6% |

The ability of neural networks to address the RSA cryptosystem has also been investigated. In a previous work of ours, we have tried to approximate the $\phi(N)$ mapping, $N \mapsto \phi(N)$, with input patterns being numbers $N = p \times q$, where $p$ and $q$ are primes, and as target patterns the $\phi(N) = (p-1) \times (q-1)$ numbers. In this case the normalization problem was no more an obstacle. What is really interesting in this case is the generalization performance of the networks. Clearly, the networks were able not only to adapt to the training data, but also to achieve very good results with respect to the test sets [84]. Indicative results on networks trained for the $\phi(N)$ mapping are exhibited in Table 1.8.

**Table 1.8.** Results for networks trained for the $\phi(N)$ mapping with $N = p \times q \leqslant 10^4$

| Topology | Epochs | $\mu_0$ | $\mu_{\pm 2}$ | $\mu_{\pm 5}$ | $\mu_{\pm 10}$ | $\mu_{\pm 20}$ |
|---|---|---|---|---|---|---|
| $1 - 5 - 5 - 1$ | 80000 | 3% | 15% | 35% | 65% | 90% |
| $1 - 7 - 8 - 1$ | 50000 | 6% | 20% | 50% | 70% | 100% |

The factorization problem is also viewed in a different setting. More specifically, approximating the value of the function $p^2 + q^2$, given the value of $N$, leads directly to the factorization of $N$ to its factors $p$ and $q$. Thus, we tested the ANNs on the approximation of the aforementioned function for several instances of $N$. The results for this problem are reported in Table 1.9.

**Table 1.9.** Results for the second setting of the factorization problem for $N$ ranging from 143 to 1003

| Topology | Epochs | $\mu_0$ | $\mu_{\pm 15}$ | $\mu_{\pm 20}$ | $\mu_{\pm 30}$ | $\mu_{\pm 40}$ |
|---|---|---|---|---|---|---|
| $1-15-1$ | 200000 | 35.1% | 36.8% | 42.1% | 43.8% | 45.6% |
| $1-20-1$ | 600000 | 35.1% | 43.8% | 45.6% | 52.6% | 56.2% |

Although the two settings of the factorization problem are computationally equivalent the approximation capabilities of the FNNs for this problem seem to be better for the first setting.

It is known that if a method for computing indices over finite fields is available, then the RSA cryptosystem breaks. In other words, the DLP is no easier than the factorization problem related to the RSA, which is confirmed by our experimental results.

In this study we consider only FNNs. In a future correspondence we intend to apply various other networks and learning techniques including non-monotone neural networks [11], probabilistic neural networks [122], self-organized maps [58], recurrent networks and radial basis function networks [41] among others. All data sets used in our experiments are available upon request.

### 1.4.4 Artificial Neural Networks Applied on Problems Related to Elliptic Curve Cryptography

In this section we study the performance of ANNs on the problem of computing the least significant bit of the discrete logarithm of a point over elliptic curves. The computation of the least significant bit of the discrete logarithm over elliptic curves with known odd order is important for cryptographic applications as it leads to the computation of all bits of the discrete logarithm. The results of this first attempt to address the specific problem using ANNs indicate that ANNs are able to adapt to the data presented with high accuracy, while the response of ANNs to unknown data is slightly higher than random selection. Another important finding is that ANNs require a small amount of storage for the known patterns in contrast to the storage needed for the data set itself [63].

#### Problem Formulation

For the discrete logarithm problem over elliptic curves, the following proposition is derived from the bit security of discrete logarithms over any cyclic group [101, 125].

**Proposition 1.** *Given an elliptic curve, $E$, over a finite field, $\mathbb{F}_q$, with known order $n$, and an oracle for a bit of the discrete logarithm that does not correspond to any power of 2 that divides the order $n$, then all the bits of the discrete logarithm can be computed in polynomial time.*

*Remark 1.* Currently, there is no polynomial algorithm for finding the order of an elliptic curve. Furthermore, the complexity for the computation of the discrete logarithm problem over elliptic curves with no knowledge of its order is exponential and, hence, it remains a computationally difficult task.

From Proposition 1 it is derived that in the case of an elliptic curve with odd order $n$, an oracle that gives the least significant bit of the discrete logarithm of a point over the elliptic curve leads to the computation of all bits of the discrete logarithm in polynomial time. Furthermore, prime order elliptic curves are considered more secure. Thus, our focus is on the computation of the least significant bit of the discrete logarithm of a point over elliptic curves of odd order. Complexity estimates for the computation of bits of the discrete logarithm over different fields can be found in [22, 120].

In relation to our problem, the considered Boolean function is defined as follows. Assume an elliptic curve $E(\mathbb{F}_p)$ and let $P = (x_P, y_P)$, $Q = (x_Q, y_Q)$ be two points of $E(\mathbb{F}_p)$, such that $Q = tP$, with $0 \leqslant t \leqslant (n-1)$. Define the Boolean function $f : \{0,1\}^{4\lceil \log p \rceil} \mapsto \{0,1\}$, with

$$f(x_P, y_P, x_Q, y_Q) = \mathrm{lsb}(t), \tag{1.27}$$

which has inputs the coordinates $x_P, y_P, x_Q, y_Q$, in binary representation, and outputs the least significant bit (lsb) of $t$, i.e., 1 if the least significant bit of $t$ is 1, and 0 otherwise. In general, a Boolean circuit that computes this function can be exponentially large in $\lceil \log p \rceil$ [22]. For the computation of this Boolean function we employ Artificial Neural Networks. Here, we focus on FNNs for the approximation of the Boolean function derived by elliptic curve cryptography defined in (1.27). For the general problem of the computation of a Boolean function by a FNNs, the following theorem proved in [2], supports the effectiveness of the proposed approach.

**Theorem 2.** *There is a threshold network with one hidden layer capable of computing any Boolean function.*

### Experimental Setup and Results

Training ANNs with threshold units requires the use of training methods that do not employ information about the derivatives of the error function. Furthermore, as shown in [23], analog neural networks can be more powerful than neural networks using thresholds, even for the computation of Boolean functions. Thus, we study the performance of ANNs using the hyperbolic tangent activation function of (1.19), which approximates a threshold function as $\lambda_2$ tends to infinity. In all experiments the output layer consists of two neurons, and the neuron with the highest output value determines the class in which the computed bit is classified. Thus, if the first neuron's output value is smaller than the value of the second neuron, the bit is considered to belong to Class 0, which corresponds to a "0" value of the bit, and vice versa. This setting enables us to use training methods that employ derivatives of the error function. In particular, we have studied the performance of three training algorithms each from a different category of training algorithms, namely the Resilient Back Propagation method (RPROP) [110], the Adaptive On–line Back Propagation method (AOBP) [73] and the Differential Evolution algorithm (DE) [126]. Regarding the topology of the networks, we have tested a variety of topologies with various numbers of neurons at each layer. We report only the best results obtained for each problem.

For the construction of the datasets the ECC_LIB library for elliptic curve cryptography [59] was used. The performance of ANNs was tested for three different datasets of the considered Boolean function that correspond to randomly chosen $p$'s of bit length 14, 20, and 32, respectively, where $\mathbb{F}_p$ is the finite field over which the elliptic curve is constructed. All data sets used are available upon request.

At each experiment the dataset was randomly partitioned into a training set and a test set. Two thirds of the dataset were assigned to the training set and the remaining one third comprised the test set. To evaluate the network performance, first we measure the average of the percentage of the training set over all 10 experiments, for which the network was able to correctly predict the least significant bit. Then, the network's performance is evaluated by measuring the average percentage of the test set over all experiments.

The best results, for the prescribed setting and $\lambda_2 = 1$, were obtained using the AOBP method and are reported in Tables 1.10, 1.11 and 1.12, respectively. The results indicate that for three bit lengths, ANNs are able to adapt to the training data with an average accuracy of 90%. With respect to the test sets, ANNs achieved for all three bit lengths an average accuracy of 57%, i.e. a slightly higher than random selection. Regarding the training epochs required in each case, as the bit length of $p$ increases, more epochs are needed to achieve the same accuracy.

Another interesting finding regarding the training set, is that the network is able to learn the training patterns and respond correctly about the least significant bit of the discrete logarithm, using less storage than that required by the corresponding dataset. The results for the data compression are reported in Table 1.13. In Table 1.13, "BL($p$)" denotes the bit length of $p$, "Data Stor." denotes the storage bits required for the dataset, "ANN Stor." denotes the storage bits required for the network weights and "Accuracy" corresponds to the accuracy of the network to identify the desired value for both classes.

An interesting line of further research is to study the performance of ANNs for larger values of $p$ and elliptic curves of different order, as well as, in other related problems such as the computation of the order of elliptic curves.

## 1.5 Ridge Polynomial Networks for Cryptography

Ridge Polynomial Networks (RPNs) belong to the class of ANNs that are based on product type of neurons, i.e., neurons apply their activation function over the product of the weighted inputs. RPNs exhibit several advantages compared to ANNs based on summing units. The computation of the least significant bit of the discrete logarithm is important for cryptographic applications as it is related to the computation of all bits of the discrete logarithm [40]. For this reason, in this section we relate these two aspects providing some theoretical conclusions and insights for future research.

For completeness purposes let us first introduce the Pi-Sigma networks that are the building components of the RPNs, along with some theoretical background of RPNs. A Pi-Sigma network (PSN) is a feedforward network

**Table 1.10.** Results for $p$ of bit length 14, using $56 - 3 - 2$ topology

| Epochs | | Train | | | Test | | |
|---|---|---|---|---|---|---|---|
| | | Class 0 | Class 1 | Accuracy | Class 0 | Class 1 | Accuracy |
| 500 | Class 0 | 168 | 33 | 83.58% | 30 | 24 | 55.56% |
| | Class 1 | 48 | 151 | 75.88% | 23 | 23 | 50.00% |
| 650 | Class 0 | 184 | 17 | 91.54% | 33 | 21 | 61.11% |
| | Class 1 | 32 | 167 | 83.92% | 23 | 23 | 50.00% |
| 700 | Class 0 | 183 | 18 | 91.04% | 33 | 21 | 61.11% |
| | Class 1 | 30 | 169 | 84.92% | 21 | 25 | 54.35% |
| 1000 | Class 0 | 186 | 15 | 92.54% | 33 | 21 | 61.11% |
| | Class 1 | 25 | 174 | 87.44% | 18 | 28 | 60.87% |

**Table 1.11.** Results for $p$ of bit length 20, using $80 - 3 - 2$ topology

| Epochs | | Train | | | Test | | |
|---|---|---|---|---|---|---|---|
| | | Class 0 | Class 1 | Accuracy | Class 0 | Class 1 | Accuracy |
| 2000 | Class 0 | 186 | 14 | 93.0% | 32 | 26 | 55.17% |
| | Class 1 | 23 | 177 | 88.5% | 17 | 25 | 59.52% |
| 3000 | Class 0 | 191 | 9 | 95.5% | 30 | 28 | 51.72% |
| | Class 1 | 19 | 181 | 90.5% | 21 | 21 | 50.00% |
| 4000 | Class 0 | 194 | 6 | 98.0% | 32 | 26 | 55.17% |
| | Class 1 | 18 | 182 | 91.0% | 19 | 23 | 54.76% |
| 6000 | Class 0 | 196 | 4 | 98.0% | 33 | 25 | 56.90% |
| | Class 1 | 17 | 183 | 91.5% | 20 | 22 | 52.38% |

**Table 1.12.** Results for $p$ of bit length 32, using $128-3-2$ topology

| Epochs | | Train | | | Test | | |
|---|---|---|---|---|---|---|---|
| | | Class 0 | Class 1 | Accuracy | Class 0 | Class 1 | Accuracy |
| 4000 | Class 0 | 193 | 5 | 97.47% | 36 | 21 | 63.16% |
| | Class 1 | 16 | 186 | 92.08% | 20 | 23 | 53.49% |
| 5000 | Class 0 | 193 | 5 | 97.47% | 36 | 21 | 63.16% |
| | Class 1 | 15 | 187 | 92.57% | 19 | 24 | 55.81% |
| 8000 | Class 0 | 193 | 5 | 97.47% | 35 | 22 | 61.40% |
| | Class 1 | 14 | 188 | 93.07% | 18 | 25 | 58.14% |
| 9000 | Class 0 | 193 | 5 | 97.47% | 35 | 22 | 61.40% |
| | Class 1 | 14 | 188 | 93.07% | 16 | 27 | 62.79% |

**Table 1.13.** Data compression results

| BL($p$) | Data Stor. | ANN Stor. | Accuracy |
|---|---|---|---|
| 14 | 23200 | 8400 | 89.99% |
| 20 | 32800 | 11856 | 94.75% |
| 32 | 52000 | 18768 | 95.27% |

with a single “hidden” layer of linear units that uses product units in the output layer, i.e., it uses *products of sums* of input components. The presence of only one layer of adaptive weights at PSNs results in fast training. There are two types of PSNs, the *Analog Pi-Sigma Networks* (APSNs) and the *Binary Pi-Sigma Networks* (BPSNs). A generalization of APSN, the Ridge polynomial networks (RPNs) is proved to have universal approximation capability [37]. BPSNs, on the other hand, are capable of realizing any Boolean function [119].

In Fig. 1.4 a PSN with a single output is illustrated. This network is a fully connected two-layered feedforward network. However, the summing layer is not “hidden” as in the case of the Multilayer Perceptron (MLP), since the weights from this layer to the output layer are fixed at the value 1. This property contributes to reducing the required training time.

Let $\mathbf{x} = (1, x_1, \ldots, x_N)^\top$ be an $N{+}1$-dimensional augmented input column vector, where $x_k$ denotes the $k$-th component of $\mathbf{x}$. The inputs are weighted by $K$ $(N+1)$-dimensional weight vectors $\mathbf{w}_j = (w_{0j}, w_{1j}, \ldots, w_{Nj})^\top$, $j = 1, 2, \ldots, K$ and summed by a layer of $K$ linear summing units, where $k$ is the desired order of the network. The output of the $j$th summing unit, $h_j$, is given as follows:

$$h_j = \mathbf{w}_j^\top \mathbf{x} = \sum_{k=1}^{N} w_{kj} x_k + w_{0j}, \quad j = 1, 2, \ldots, K. \tag{1.28}$$

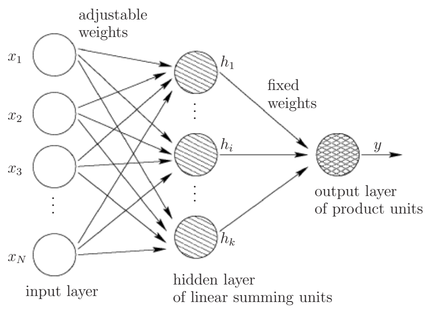

**Fig. 1.4.** A Pi-Sigma network (PSN) with one output unit

The output $y$ of the network is given by:

$$y = \sigma(\prod_{j=1}^{K} h_j) = \sigma(net), \tag{1.29}$$

where $\sigma(\cdot)$ is a suitable activation function and $net = \prod_{j=1}^{K} h_j$. In the above, $w_{kj}$ is an adjustable weight from input $x_k$ to the $j$th summing unit and $w_{0j}$ is the threshold of the $j$th summing unit. Weights can assume arbitrary real values.

The network shown in Fig. 1.4 is called a $K$-th order PSN since $K$ summing units are incorporated. The total number of adjustable weight connections, including the adjustable thresholds, for a $K$-th order PSN with $N$-dimensional inputs is $(N+1)K$. If multiple outputs are required, an independent summing layer is needed for each output. Thus, for an $M$-dimensional output vector $\mathbf{y}$, a total of $\sum_{i=1}^{M}(N+1)K_i$ adjustable weight connections are present, where $K_i$ is the number of summing units for the $i$th output. This enables the network to be incrementally expandable, since the order can be increased by adding another summing unit and associated weights, without disturbing any connection previously established. PSNs can handle both analog and binary input/output by using a suitable nonlinear activation function $\sigma(\cdot)$.

Regarding the approximation capabilities of PSNs, although the activation function is applied on a $K$-th order polynomial when $K$ summing units are used and the exponents $i_j$ sum up to $K$, this does not mean that only $K$-th order terms can be used, since by letting an extra input (the bias) be fixed at value 1, terms of order less than $K$ are also realized. This $K$-th order polynomial, however, does not have full degrees of freedom since the coefficients are composed of sums and products of $w_{kj}$s and thus are not independent. Thus, a PSN cannot uniformly approximate all continuous multivariate functions that can be defined on a compact set. However, the theory of ridge polynomials can be used to show that universal approximation capability can be achieved simply by summing the outputs of APSNs of different orders. The resulting network is a generalization of PSN, which is called *Ridge Polynomial Network* (RPN), and is developed as follows.

For $\mathbf{x} = (x_1, \ldots, x_N)^\top$ and $\mathbf{w} = (w_1, \ldots, w_N)^\top \in \mathbb{R}^N$, we denote as $\langle \mathbf{x}, \mathbf{w} \rangle$ their inner product, i.e., $\langle \mathbf{x}, \mathbf{w} \rangle = \sum_{i=1}^{N} x_i w_i$. For a given compact set $C \subset \mathbb{R}^N$, all functions defined on $C$ in the form of $f(\langle \mathbf{x}, \mathbf{w} \rangle)$, where $f$ is a continuous function in one variable, are called ridge functions. A *ridge polynomial* is a ridge function that can be represented as

$$\sum_{i=0}^{n} \sum_{j=1}^{m} a_{ij} \langle \mathbf{x}, \mathbf{w} \rangle^i, \tag{1.30}$$

for some $a_{ij} \in \mathbb{R}$ and $\mathbf{w}_{ij} \in \mathbb{R}^N$.

It was proved in [15,16] that any polynomial in $\mathbb{R}^N$ with degree less than or equal to $k$ can be represented by a ridge polynomial and, furthermore, it can be realized by an RPN [37]. From these results and the Stone-Weierstrass theorem the uniform approximation capability of the ridge polynomials of (1.30) is implied [37]. For applications of the RPNs in approximation and root finding see [31].

The ridge polynomial network (RPN), is defined as a feedforward network based on the generalized form of ridge polynomials [37]:

$$p(\mathbf{x}) = \sum_{j=1}^{n_{\text{total}}} \prod_{i=1}^{j} (\langle \mathbf{x}, \mathbf{w}_{ji} \rangle + w_{ij}), \tag{1.31}$$

where $n_{\text{total}} = \sum_{l=0}^{k} n_l$, and approximates an unknown function $f$ on a compact set $C \subset \mathbb{R}^N$ as

$$\begin{aligned} f(\mathbf{x}) \approx (\langle \mathbf{x}, \mathbf{w}_{11} \rangle + w_{11}) + (\langle \mathbf{x}, \mathbf{w}_{21} \rangle + w_{21})(\langle \mathbf{x}, \mathbf{w}_{22} \rangle + w_{22}) + \cdots \\ + (\langle \mathbf{x}, \mathbf{w}_{N1} \rangle + w_{N1}) \cdots (\langle \mathbf{x}, \mathbf{w}_{NN} \rangle + w_{NN}). \end{aligned} \tag{1.32}$$

Each product term in (1.32) can be obtained as the output of a PSN with linear output units. Thus, the formulation of RPNs can be considered as a generalization of PSNs. Figure 1.5 represents a generic network architecture of the RPN using PSNs as building blocks. The RPN has only a single layer of adjustable weights which is beneficial in terms of training speed. Note that (1.32) serves as the basis for an incremental learning algorithm where PSNs of successively higher orders can be added until a desirable level of accuracy is obtained.

The uniform approximation capability of RPNs, their faster training compared to other kinds of ANNs, and the ability to perform incremental training, renders them a promising methodology for application to cryptological problems. Some conclusions derived by relating these two fields follow, providing an insight for future research.

Considering the computation of the rightmost bit of the discrete logarithm by real polynomials, the following theorem holds:

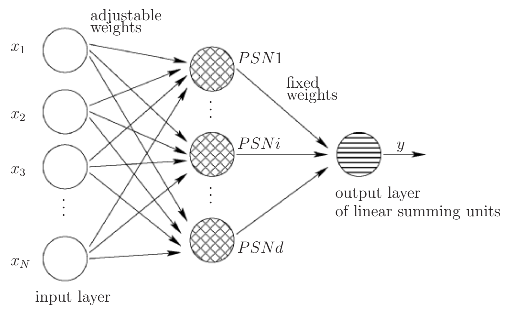

**Fig. 1.5.** A ridge polynomial network (RPN) with one linear output unit

**Theorem 3 ([120]).** *Let* $0 \leqslant M < M+H \leqslant p-1$. *Assume that a polynomial* $f(X) \in \mathbb{R}[X]$ *is such that* $f(x) \geqslant 0$ *if* $x$ *is a quadratic residue modulo* $p$ *and* $f(x) < 0$ *otherwise, for every element* $x \in S$ *from some set* $S \subseteq \{M+1, \ldots, M+H\}$ *of cardinality* $|S| \geqslant H-s$. *Then for any* $\varepsilon > 0$ *the bound*

$$\deg f \geqslant \begin{cases} H/2 - 2s - 1 - p^{1/2}\log p, & \textit{for any} \quad H, \\ \mathcal{C}(\varepsilon)\min\{H, H^2 p^{-1/2}\} - 2s - 1, & \textit{if} \quad p^{1/4+\varepsilon} \leqslant H \leqslant p^{1/2+\varepsilon}, \\ (p-1)/2 - 2s - 1, & \textit{if} \quad N = 0, H = p-1, \end{cases} \tag{1.33}$$

*holds, where* $\mathcal{C}(\varepsilon) > 0$ *depends only on* $\varepsilon$.

Moreover, regarding real multivariate polynomials that compute the rightmost bit of the discrete logarithm, the following theorem holds:

**Theorem 4 ([120]).** *Let* $\alpha_0, \alpha_1$ *be two distinct real numbers,* $r = \lfloor \log p \rfloor$, *and let a polynomial* $f(X_1, \ldots, X_r) \in \mathbb{R}[X_1, \ldots, X_r]$ *be such that* $f(\alpha_{u_1}, \ldots, \alpha_{u_r}) \geqslant 0$, *if* $x$ *is a quadratic residue modulo* $p$ *and* $f(\alpha_{u_1}, \ldots, \alpha_{u_r}) < 0$, *otherwise, where* $x = u_1 \ldots u_r$ *is the bit representation of* $x$, $1 \leqslant x \leqslant 2^r - 1$. *Then* $f$ *is of degree* $\deg f \geqslant \log r + o(\log r)$ *and contains at least* $\operatorname{spr} f \geqslant 0.25r + o(r)$ *distinct monomials.*

Since, the universal approximation capability of RPNs is achieved by summing the outputs of PSNs of different orders, for the approximation of the real polynomials that compute the rightmost bit of the discrete logarithm the following corollary is derived.

**Corollary 1.** *Assume the polynomial* $f(X) \in \mathbb{R}[X]$ *that satisfies the conditions of Theorem 3, i.e., it is such that* $f(x) \geqslant 0$ *if* $x$ *is a quadratic residue modulo* $p$ *and* $f(x) < 0$ *otherwise, for every element* $x \in S$ *from some set* $S \subseteq \{M+1, \ldots, M+H\}$ *of cardinality* $|S| \geqslant H - s$, *and the polynomial* $f(X_1, \ldots, X_r)$ *that satisfies the conditions of Theorem 4, i.e., it is such that* $f(\alpha_{u_1}, \ldots, \alpha_{u_r}) \geqslant 0$, *if* $x$ *is a quadratic residue modulo* $p$ *and* $f(\alpha_{u_1}, \ldots, \alpha_{u_r}) < 0$, *otherwise, where* $x = u_1 \ldots u_r$ *is the bit representation of* $x$, $1 \leqslant x \leqslant 2^r - 1$. *Then, the following conclusions hold.*

(*a*) *There exist two RPNs that realize the polynomial $f(X)$ and the polynomial $f(X_1, \ldots, X_r)$, respectively, and*
(*b*) *The maximum orders of the nodes comprising each RPN, are required to be for the first case equal to*

$$\deg f(X) \geqslant \begin{cases} H/2 - 2s - 1 - p^{1/2}\log p, & \textit{for any} \quad H, \\ \mathcal{C}(\varepsilon)\min\{H, H^2 p^{-1/2}\} - 2s - 1, & \textit{if} \quad p^{1/4+\varepsilon} \leqslant H \leqslant p^{1/2+\varepsilon}, \\ (p-1)/2 - 2s - 1, & \textit{if} \quad N = 0, H = p - 1, \end{cases} \tag{1.34}$$

*where $\mathcal{C}(\varepsilon) > 0$ depends only on $\varepsilon$, and equal to*

$$\deg f(X_1, \ldots, X_r) \geqslant \log r + o(\log r), \tag{1.35}$$

*for the second case.*

Additionally, the inherent ability of RPNs to expand the final polynomial that they realize after training, as an ordinary multivariate polynomial, may prove of major importance in the field of cryptography. Finally, relating the fact that many cryptographic results consider the use of Boolean functions with the capability of BPSNs to realize any Boolean function, can provide theoretical conclusions for this class of functions too.

## 1.6 Summary

The past decade has witnessed an increasing interest in the application of Computational Intelligence (CI) methods to problems derived from the field of cryptography and cryptanalysis. This is not only due to the effectiveness of these methods observed in many other scientific fields but also due to the need for automated techniques in the design and cryptanalysis of cryptosystems.

In this contribution, a brief review to cryptography and CI methods is initially provided. Then, a short survey of the applications of CI to cryptographic problems follows, and our contribution in this field is analytically presented. More specifically, at first three cryptographic problems derived from classical public key cryptosystems are formulated as discrete optimization tasks and Evolutionary Computation (EC) methods are applied to address them. Next, EC methods for the partial cryptanalysis of a Feistel cipher, the Data Encryption Standard reduced to four and six rounds, respectively, are considered. The effectiveness of Artificial Neural Networks (ANNs) for classical cryptographic problems and problems related to elliptic curve cryptography, follow. Lastly, some theoretical results are derived based on the composition of a specific class of ANNs, namely the Ridge Polynomial Networks, with theoretical issues of cryptography.

The experimental results presented for each considered problem suggest that problem formulation and representation are critical determinants of the performance of CI methods in cryptography. Regarding the application of EC methods in cryptanalysis, the proper definition of the fitness function such that no deceptive landscapes are created, is of major importance. Furthermore, the performance of ANNs in cryptographic problems depends on the problem formulation and data representation. A second conclusion derived is that EC methods (and CI methods in general) can be used as a practical assessment for the efficiency and the effectiveness of proposed cryptographic systems, meaning that they can "sense" flawed cryptographic schemes by finding patterns before more complex methods are employed for their analysis.

## References

1. Adleman L (1979) A subexponential algorithm for discrete logarithm problem with applications to cryptography. In: Proceedings of the 20th FOCS, pp. 55–60
2. Anthony M (2003) Boolean functions and artificial neural networks. Technical report, CDAM, The London School of Economics and Political Science. CDAM Research Report LSE-CDAM-2003-01
3. Bäck T (1996) Evolutionary Algorithms in Theory and Practice : Evolution Strategies, Evolutionary Programming, Genetic Algorithms. Oxford University Press
4. Bagnall T, McKeown G. P, Rayward-Smith V. J (1997) The cryptanalysis of a three rotor machine using a genetic algorithm. In: Bäck T (ed) Proceedings of the 7th International Conference on Genetic Algorithms (ICGA97), San Francisco, CA, Morgan Kaufmann
5. Barbieri A, Cagnoni S, Colavolpe G (2004) A genetic approach for generating good linear block error-correcting codes. Lecture Notes in Computer Science 3103:1301–1302
6. Biham E, Shamir A (1991) Differential cryptanalysis of DES–like cryptosystems. Journal of Cryptology
7. Biham E, Shamir A (1993) Differential Cryptanalysis of the Data Encryption Standard. Springer–Verlag
8. Blake I (1999) Elliptic Curves in Cryptography. London Mathematical Society Lecture Notes Series vol. 265. Cambridge University Press
9. Blum A, Furst M, Kearns M, Lipton R. J (1994) Cryptographic primitives based on hard learning problems. Lecture Notes in Computer Science 773:278–291
10. Bonabeau E, Dorigo M, Théraulaz G (1999) From Natural to Artificial Swarm Intelligence. Oxford University Press, New York
11. Boutsinas B, Vrahatis M. N (2001) Artificial nonmonotonic neural networks. Artificial Intelligence 132:1–38
12. Burnett L, Carter G, Dawson E, Millan W (2001) Efficient methods for generating Mars-like S-boxes. Lecture Notes in Computer Science 1978(4):300–313
13. Carrol J, Martin S (1986) The automated cryptanalysis of substitution ciphers. Cryptologia 10(4):193–209
14. Chang Y.-C, Lu C.-J (2001) Oblivious polynomial evaluation and oblivious neural learning. Lecture Notes in Computer Science 2248:369–384

15. Chui C.K, Li X (1991) Realization of neural networks with one hidden layer. Technical report, Center for Approximation Theory, Dept. of Mathematics, Texas A&M University
16. Chui C.K, and Li X (1992) Approximation by Ridge functions and neural networks with one hidden layer. Journal of Approximation Theory 70:131–141
17. Clark A (1998) Optimisation Heuristics for Cryptography. PhD Thesis, Queensland University of Technology, Australia
18. Clark J.A, Jacob J.L (2000) Two-stage optimisation in the design of Boolean functions. Lecture Notes in Computer Science 1841:242–254
19. Clark J.A, Jacob J.L (2002) Fault injection and a timing channel on an analysis technique. Lecture Notes in Computer Science 2332:181–196
20. Clark J.A, Jacob J.L, Stepney S (2004) The design of S-boxes by Simulated Annealing. In: CEC 2004: International Conference on Evolutionary Computation, Portland OR, USA, June 2004, pp. 1517–1524. IEEE
21. Clerc M, Kennedy J (2002) The particle swarm–explosion, stability, and convergence in a multidimensional complex space. IEEE Transactions on Evolutionary Computation 6(1):58–73
22. Coppersmith D, Shparlinski I (2000) On polynomial approximation of the discrete logarithm and the Diffie–Hellman mapping. Journal of Cryptology 13:339–360
23. DasGupta B, Schnitger G (1996) Analog versus discrete neural networks. Neural Computation 8(4):805–818
24. De Jong K.A (1985) Genetic algorithms: A 10 year perspective. In: Proceedings of the First International Conference on Genetic Algorithms pp. 169–177. Lawrence Erlbaum Associates.
25. Diffie W, Hellman M.E (1976) New directions in cryptography. IEEE Transactions on Information Theory IT-22(6):644–654
26. Dontas K, Jong K (1990) Discovery of maximal distance codes using genetic algorithms. In: Proceedings of the Second International IEEE Conference on Tools for Artificial Intelligence pp. 805–811
27. Dorigo M, Gambardella L.M (1997) Ant colonies for the traveling salesman problem. BioSystems 43:73–81
28. Eberhart R.C, Simpson P, Dobbins R (1996) Computational Intelligence PC Tools. Academic Press
29. ElGamal T (1985) A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Transactions on Information Theory 31(4):469–472
30. Engelbrecht A (2002) Computational Intelligence: An Introduction. John Wiley & Sons
31. Epitropakis M.G, Vrahatis M.N (2005) Root finding and approximation approaches through neural networks. SIGSAM Bulletin: Communications in Computer Algebra, ACM Press 39(4):118–121
32. Feistel H (1973) Cryptography and computer privacy. Scientific American
33. Fogel D.B (1993) Evolving behaviours in the iterated prisoner's dilemma. Evolutionary Computation 1(1):77–97
34. Fogel D.B (1995) Evolutionary Computation: Towards a New Philosophy of Machine Intelligence. IEEE Press, Piscataway, NJ
35. Fogel D.B, Owens A.J, Walsh M.J (1966) Artificial Intelligence Through Simulated Evolution. John Wiley, Chichester, UK
36. Forsyth W.S, Safavi-Naini R (1993) Automated cryptanalysis of substitution ciphers. Cryptologia 17(4):407–418

37. Ghosh J, Shin Y (1992) Efficient higher-order neural networks for classification and function approximation. International Journal of Neural Systems 3:323–350
38. Goldberg D.E (1989) Genetic Algorithms in Search, Optimization, and Machine Learning. Addison Wesley, Reading, MA
39. Hassoun M.H (1995) Foundamentals of Artificial Neural Networks. MIT Press, Cambridge, MA
40. Håstad J, Näslund M (2004) The security of all RSA and discrete log bits. Journal of the ACM 51(2):187–230
41. Haykin S (1999) Neural Networks, A Comprehensive Foundation. Prentice Hall, New Jersey, USA, 2nd edition edition
42. Herdy M (1991) Application of the evolution strategy to discrete optimization problems. Lecture Notes in Computer Science pp. 188–192
43. Hernández J, Isasi P, Ribagorda A (2002) An application of genetic algorithms to the cryptoanalysis of one round TEA. In: Proc. of the 2002 Symposium on Artificial Intelligence and its Application
44. Hernández J, Sierra J, Isasi P, Ribagorda A (2002) Genetic cryptoanalysis of two rounds TEA. Lecture Notes in Computer Science 2331:1024–1031
45. Holland J.H (1975) Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor
46. Hornik K (1989) Multilayer feedforward networks are universal approximators. Neural Networks 2:359–366
47. Hunter D, McKenzie A (1983) Experiments with relaxation algorithms for breaking simple substitution ciphers. The Computer Journal 26(1):68–71
48. Isasi P, Hernández J (2004) Introduction to the applications of evolutionary computation in computer security and cryptography. Computational Intelligence 20(3):445–449
49. Jakobsen T (1995) A fast method for cryptanalysis of substitution ciphers. Cryptologia 19(3):265–274
50. Karras D, Zorkadis V (2002) Strong pseudorandom bit sequence generators using neural network techniques and their evaluation for secure communications. Lecture Notes in Artificial Intelligence 2557:615–626
51. Karras D, Zorkadis V (2003) On neural network techniques in the secure management of communication systems through improving and quality assessing pseudorandom stream generators. Neural Networks 16:899–905
52. Kennedy J, Eberhart R.C (2001) Swarm Intelligence. Morgan Kaufmann Publishers
53. King J, Bahler D (1992) An implementation of probabilistic relaxation in the cryptanalysis of simple substitution ciphers. Cryptologia 16(3):215–225
54. Kinzel W, Kanter I (2002) Interacting neural networks and cryptography. In: Kramer B (ed) Advances in Solid State Physics. vol. 42, pp. 383–391. Springer–Verlag
55. Klimov A, Mityagin A, Shamir A (2002) Analysis of neural cryptography. Lecture Notes in Computer Science 2501:288–298
56. Knudsen L.R, Meier W (1999) A new identification scheme based on the perceptrons problem. Lecture Notes in Computer Science 1592:363–374
57. Koblitz N (1987) Elliptic curve cryptosystems. Mathematics of Computation 48:203–209
58. Kohonen T (2000) Self-Organizing Maps. Springer-Verlag, Berlin, 3rd edition

59. Konstantinou E, Stamatiou Y, Zaroliagis C (2002) A software library for elliptic curve cryptography. Lecture Notes in Computer Science 2461:625–637
60. Kotlarz P, Kotulski Z (2005) On application of neural networks for s-boxes design. Lecture Notes in Artificial Intelligence 3528:243–248
61. Koza J.R (1992) Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge, MA
62. Lange T, Winterhof A (2002) Incomplete character sums over finite fields and their application to the interpolation of the discrete logarithm by boolean functions. Acta Arithmetica 101(3):223–229
63. Laskari E.C, Meletiou G.C, Stamatiou Y.C, Tasoulis D.K, Vrahatis M.N (2006) Assessing the effectiveness of artificial neural networks on problems related to elliptic curve cryptography. Mathematical and Computer Modelling. to appear
64. Laskari E.C, Meletiou G.C, Stamatiou Y.C, Vrahatis M.N (2005) Evolutionary computation based cryptanalysis: A first study. Nonlinear Analysis: Theory, Methods and Applications 63:e823–e830
65. Laskari E.C, Meletiou G.C, Vrahatis M.N (2005) Problems of Cryptography as Discrete Optimization Tasks. Nonlinear Analysis: Theory, Methods and Applications 63:e831–e837
66. Laskari E.C, Meletiou G.C, Stamatiou Y.C, Vrahatis M.N (2006) Applying evolutionary computation methods for the cryptanalysis of Feistel ciphers. Applied Mathematics and Computation. to appear
67. Laskari E.C, Meletiou G.C, Tasoulis D.K, Vrahatis M.N (2005) Aitken and Neville inverse interpolation methods over finite fields. Applied Numerical Analysis and Computational Mathematics 2(1):100–107
68. Laskari E.C, Meletiou G.C, Tasoulis D.K, Vrahatis M.N (2005) Transformations of two cryptographic problems in terms of matrices. SIGSAM Bulletin: Communications in Computer Algebra, ACM Press 39(4):127–130
69. Laskari E.C, Meletiou G.C, Tasoulis D.K, Vrahatis M.N (2006) Studying the performance of artificial neural networks on problems related to cryptography. Nonlinear Analysis Series B: Real World Applications 7(5):937–942
70. Laskari E.C, Meletiou G.C, Vrahatis M.N (2004) The discrete logarithm problem as an optimization task: A first study. In: Proceedings of the IASTED International Conference on Artificial Intelligence and Applications pp. 1–6. ACTA Press
71. Laskari E.C, Parsopoulos K.E, Vrahatis M.N (2002) Particle swarm optimization for integer programming. In: Proceedings of the IEEE 2002 Congress on Evolutionary Computation pp. 1576–15812. IEEE Press.
72. Laskari E.C, Parsopoulos K.E, Vrahatis M.N (2002) Particle swarm optimization for minimax problems. In: Proceedings of the IEEE 2002 Congress on Evolutionary Computation pp. 1576–1581. IEEE Press.
73. Magoulas G.D, Plagianakos V.P, Vrahatis M.N (2001) Adaptive stepsize algorithms for on-line training of neural networks. Nonlinear Analysis T.M.A. 47(5):3425–3430
74. Magoulas G.D, Vrahatis M.N (2006) Adaptive algorithms for neural network supervised learning: a deterministic optimization approach. International Journal of Bifurcation and Chaos 16(7):1929–1950
75. Magoulas G.D, Vrahatis M.N, Androulakis G.S (1997) Effective backpropagation training with variable stepsize. Neural Networks 10(1):69–82

76. Magoulas G.D, Vrahatis M.N, Androulakis G.S (1999) Increasing the convergence rate of the error backpropagation algorithm by learning rate adaptation methods. Neural Computation 11(7):1769–1796
77. Mathews R (1993) The use of genetic algorithms in cryptanalysis. Cryptologia 17(4):187–201
78. Matsui M (1994) Linear cryptanalysis method for DES cipher. Lecture Notes in Computer Science 765:386–397
79. Matsui M, Yamagishi A (1992) new method for known plaintext attack of feal cipher. Lecture Notes in Computer Science pp. 81–91
80. Maurer U, Wolf S (1999) The relationship between breaking the diffie-hellman protocol and computing discrete logarithms. SIAM Journal on Computing 28:1689–1721
81. Meletiou G.C (1992) A polynomial representation for exponents in $\mathbb{Z}_p$. Bulletin of the Greek Mathematical Society 34:59–63
82. Meletiou G.C (1993) Explicit form for the discrete logarithm over the field GF$(p, k)$. Archivum Mathematicum (Brno) 29(1–2):25–28
83. Meletiou G.C, Mullen G.L (1992) A note on discrete logarithms in finite fields. Applicable Algebra in Engineering, Communication and Computing 3(1):75–79
84. Meletiou G.C, Tasoulis D.K, Vrahatis M.N (2003) Cryptography through interpolation approximation and computational inteligence methods. Bulletin of the Greek Mathematical Society 48:61–75
85. Menezes A, van Oorschot P, Vanstone S (1996) Handbook of applied cryptography. CRC Press series on discrete mathematics and its applications. CRC Press
86. Merkle R.C, Hellman M.E (1978) Hiding information and signatures in trapdoor knapsacks. IEEE Transactions on Information Theory 24:525–530
87. Michalewicz Z (1994) Genetic Algorithms + Data Structures = Evolution Programs. Springer, Berlin
88. Millan W, Clark A, Dawson E (1997) Smart hill climbing finds better boolean functions. In: Proceedings of the 4th Workshop on Selected Areas in Cryptography
89. Millan W, Clark A, Dawson E (1999) Boolean function design using hill climbing methods. Lecture Notes in Computer Science 1587:1–11
90. Miller V (1986) Uses of elliptic curves in cryptography. Lecture Notes in Computer Science 218:417–426
91. Mislovaty R, Perchenok Y, Kanter I, Kinzel W (2002) Secure key-exchange protocol with an absence of injective functions. Phys. Rev. E 66(6):066102–1–066102–5
92. Møller M.F (1993) A scaled conjugate gradient algorithm for fast supervised learning. Neural Networks 6:525-533
93. Mullen G.L, White D (1986) A polynomial representation for logarithms in GF$(q)$. Acta Arithmetica 47:255–261
94. National Bureau of Standards, U.S. Department of Commerce, FIPS pub. 46. Data Encryption Standard. January 1977.
95. Niederreiter H (1990) A short proof for explicit formulas for discrete logarithms in finite fields. Applicable Algebra in Engineering, Communication and Computing 1:55–57
96. Odlyzko A (2000) Discrete logarithms: The past and the future. Designs, Codes, and Cryptography 19(2–3):129–145

97. Parsopoulos K.E, Vrahatis M.N (2002) Initializing the particle swarm optimizer using the nonlinear simplex method. In: Grmela A, Mastorakis N.E (eds) Advances in Intelligent Systems, Fuzzy Systems, Evolutionary Computation pp. 216–221
98. Parsopoulos K.E, Vrahatis M.N (2002) Recent approaches to global optimization problems through particle swarm optimization. Natural Computing 1(2–3):235–306
99. Parsopoulos K.E, Vrahatis, M.N (2004) On the computation of all global minimizers through particle swarm optimization. IEEE Transactions on Evolutionary Computation 8(3):211–224
100. Peleg S, Rosenfeld A (1979) Breaking substitution ciphers using a relaxation algorithm. Communications of the ACM 22(11):598–605
101. Peralta R (1986) Simultaneous security of bits in the discrete log. Lecture Notes in Computer Science 219:62–72
102. Pincus A (1999) Approximation theory of the mlp model in neural networks. Acta Numerica pp. 143–195
103. Plagianakos V, Vrahatis M.N (2002) Parallel Evolutionary Training Algorithms for “Hardware-Friendly” Neural Networks. Natural Computing 1:307–322
104. Pohlig S.C, Hellman M (1978) An improved algorithm for computing logarithms over GF($p$) and its cryptographic significance. IEEE Transactions on Information Theory 24:106–110
105. Pointcheval D (1994) Neural networks and their cryptographic applications. In: Charpin P (ed) INRIA, Livres de resumes EUROCODE’94
106. Pointcheval D (1995) A new identification scheme based on the perceptrons problem. Lecture Notes in Computer Science 950:318–328
107. Ramzan Z (1998) On Using Neural Networks to Break Cryptosystems. PhD Thesis. Laboratory of Computer Science, MIT
108. Rao S.S (1996) Engineering Optimization–Theory and Practice. Wiley Eastern, New Delhi
109. Rechenberg I (1973) Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. Frommann-Holzboog Verlag, Stuttgart, Germany
110. Riedmiller M, Braun H (1993) A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In: Proceedings of the IEEE International Conference on Neural Networks pp. 586–591
111. Rivest R (1991) Cryptography and machine learning. Lecture Notes in Computer Science 739:427–439
112. Rivest R, Shamir A, Adleman L (1978) A method for obtaining digital signatures and public key cryptosystems. Communications of the ACM 21:120–126
113. Rosen-Zvi M, Kanter I, Kinzel W (2002) Cryptography based on neural networks – analytical results. Journal of Physics A: Mathematical and General 35(47):L707–L713
114. Rumelhart D, Hinton G, Williams R (1986) Learning internal representations by error propagation. In: RumelhartD.E, McClelland J.L (eds) Parallel distributed processing: Explorations in the microstructure of cognition. vol. 1 pp. 318–362. MIT Press
115. Ruttor A, Kinzel W, Kanter I (2005) Neural cryptography with queries. Journal of Statistical Mechanics pp. P01009
116. Ruttor A, Kinzel W, Shacham L, Kanter I (2004) Neural cryptography with feedback. Physical Review E 69(4):046110–1–046110–7

117. Schwefel H.-P (1995) Evolution and Optimum Seeking. Wiley, New York
118. Shi Y, Eberhart R.C (1998) A modified particle swarm optimizer. In: Proceedings of the IEEE Conference on Evolutionary Computation. Anchorage, AK
119. Shin Y, Ghosh J (1991) Realization of Boolean functions using binary Pi-Sigma networks. In: Proceedings of the Conference on Artificial Neural Networks in Engineering. St. Louis
120. Shparlinski I (ed) (2003) Cryptographic Applications of Analytic Number Theory. Progress in Computer Science and Applied Logic. Birkhäuser Verlag
121. Silverman J.H (1986) The Arithmetic of Elliptic Curves. Springer-Verlag
122. Specht D.F (1990) Probabilistic neural networks. Neural Networks 3(1):109–118
123. Spillman R (1993) Cryptanalysis of knapsack ciphers using genetic algorithms. Cryptologia 17(4):367–377
124. Spillman R, Janssen M, Nelson B, Kepner M (1993) Use of a genetic algorithm in the cryptanalysis of simple substitution ciphers. Cryptologia 17(1):31–44
125. Stinson D (1995) Cryptography: Theory and Practice (Discrete Mathematics and Its Applications). CRC Press
126. Storn R, Price K (1997) Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. Journal of Global Optimization 11:341–359
127. Tasoulis D.K, Pavlidis N.G, Plagianakos V.P, Vrahatis M.N (2004) Parallel Differential Evolution. In: Proceedings of the IEEE 2004 Congress on Evolutionary Computation (CEC 2004), Portland
128. Terano T, Asai K, Sugeno M (1992) A Complete Introduction to the Field: Fuzzy Systems Theory and Its Applications. Academic Press
129. Vertan C, Geangala C (1996) Breaking the Merkle-Hellman cryptosystem by genetic algorithms: Locality versus performance. In: Zimmermann H, Negoita M, Dascalu D (eds), Real World Applications of Intelligent Technologies pp. 201–208. Editura Academiei Romanie, Bucharest
130. Vrahatis M.N, Androulakis G.S, Lambrinos J.N, Magoulas G.D (2000) A class of gradient unconstrained minimization algorithms with adaptive stepsize. Journal of Computational and Applied Mathematics 114(2):367–386
131. White H (1990) Connectionist nonparametric regression: Multilayer feedforward networks can learn arbitrary mappings. Neural Networks 3:535–549
132. Wilson D, Martinez T (1997) Improved heterogeneous distance functions. Journal of Artificial Intelligence Research 6:1–34
133. Winterhof A (2001) A note on the interpolation of the Diffie-Hellman mapping. Bulletin of Australian Mathematical Society 64(3):475–477
134. Winterhof A (2002) Polynomial interpolation of the discrete logarithm. Designs, Codes and Cryptography 25(1):63–72
135. Yue T.-W, Chiang S (2001) The general neural-network paradigm for visual cryptography. Lecture Notes in Computer Science 2084:196–206