On the Optimal Computation of Finite Field Exponentiation

Nareli Cruz-Cortés, Francisco Rodríguez-Henríquez and Carlos Coello Coello

Computer Science Section, Electrical Engineering Department
Centro de Investigación y de Estudios Avanzados del IPN
Av. Instituto Politécnico Nacional No. 2508, México D.F.
nareli@computacion.cs.cinvestav.mx, {francisco, coello}@cs.cinvestav.mx

Abstract. It has been shown that the optimal computation of finite field exponentiation is closely related to the problem of finding a suitable addition chain with the shortest possible length. However, obtaining the shortest addition chain for a given arbitrary exponent is an **NP**-hard problem. Hence in general, we are forced to use some kind of heuristic in order to compute field exponentiation with a semi-optimal number of underlying arithmetic operations. In this paper we present a novel heuristic for that problem which is based on an immune artificial system strategy. The results obtained by our scheme yield the shortest reported lengths for the exponents typically used when computing field multiplicative inverses for error-correcting and elliptic curve cryptographic applications.

1 Introduction

The problem of efficiently compute finite field or group exponentiation is an illustrious mathematical problem with a long and interesting history related to the theoretical optimization problems found in computer science [1]. In addition to its theoretical relevance, field exponentiation has many important practical applications in the areas of error-correcting codes and cryptography. For instance, field or group exponentiation is used in several major public-key cryptosystems such as RSA, Diffie-Hellman and DSA [2].

Let α be an arbitrary element of a finite field F, an e and arbitrary positive integer. Then, field exponentiation is defined as the problem of finding an element $\beta \in F$ such that the equation $\beta = \alpha^e$ holds. In general one can follow two strategies to obtain an efficient implementation of field exponentiation. One approach is to implement field multiplication, the main building block required for field exponentiation, as efficiently as possible. The other is to reduce the total number of multiplications needed to compute β. In this paper we address the latter approach, assuming that arbitrary choices of the base element α are allowed but with the restriction that the exponent e is fixed.

There are a large number of reported algorithms to solve field exponentiation. Reported strategies include: binary, m-ary, adaptive m-ary, power tree and the factor method, to mention just a few [3, 1, 2]. All those algorithms have in common that they strive to keep the number of required field multiplications

as low as possible through the usage of a particular heuristic. However, none of those strategies can be considered to yield an optimal solution for every possible scenario and/or application. On the other hand, all the aforementioned heuristics can be mathematically described by using the concept of *addition chains*. Indeed, taking advantage of the fact that the exponents are additive, the problem of computing powers of the base element α, can be directly translated to an addition calculation. This observation leads us to the concept of an *addition chain* for a given exponent e that can be informally defined as follows.

An *addition chain* for e of length l is a sequence U of positive integers, $u_0 = 1, u_1 \ldots, u_l = e$ such that for each $i > 1$, $u_i = u_j + u_k$ for some j and k with $0 \leq j \leq k < i$.

The solely purpose in life of an addition chain is to minimize the number of multiplications required for an exponentiation. Indeed, if U is an addition chain that computes e as mentioned above then for each $\alpha \in F$ we can find $\beta = \alpha^e$ by successively computing: $\alpha, \alpha^{u_1}, \ldots, \alpha^{u_{l-1}}, \alpha^e$.

Let $l(e)$ be the shortest length of any valid addition chain for a given positive integer e. Then the theoretical minimum number of field multiplications required for computing the field operation $\beta = \alpha^e$ is precisely $l(e)$. Unfortunately, the problem of finding an addition chain for e with the shortest length $l(e)$ is an **NP**-hard problem [2]. In order to solve that optimization problem we present in this work an approach based on the Artificial Immune System (AIS) paradigm.

AIS is a relatively new computational intelligence paradigm which borrows ideas from the natural immune system to solve engineering problems. AIS has been successfully applied to solve problems in different areas such as computer and network security, scheduling, machine learning [4, 5] and optimization. Reported optimization problems solved using AIS systems include multimodal, numerical [6], and combinatorial optimization [7]. However to the best of our knowledge, the algorithm presented in this work constitutes the first attempt to use AIS as an heuristic to find optimal addition chains for field exponentiation computations. As a specific design example, we describe how to use our technique for computing field inversion via Fermat's little theorem [8, 9].

The rest of this paper is organized as follows. In Section 2 we describe the strategy followed in this paper in order to find optimal addition chains using an AIS approach. Then, in section 3 we explain how an optimal addition chain can be used in practice to solve finite field multiplicative inversion. A comparison between our results and other reported works is given in Section 4. Finally, in Section 5 some conclusions remarks are drawn.

2 Obtaining optimal Addition Chains using an AIS Approach

In this section we indicate how the AIS paradigm can be used to solve the problem of finding optimal addition chains. We start this section with a formal definition of an addition chain followed by a brief description of artificial immune systems. We end this section giving a description of the algorithm utilized.

2.1 Definition

An *addition chain* [2] U for a positive integer e of length l is a sequence of positive integers $U = \{u_0, u_1, \cdots, u_l\}$, and an associated sequence of r pairs $V = \{(v_1, v_2 \cdots, v_l\}$ with $v_i = (i_1, i_2), 0 \leq i_2 \leq i_1 < i$, such that:

- $u_0 = 1$ and $u_l = e$;
- for each $u_i, 1 \leq i \leq l, u_i = u_{i_1} + u_{i_2}$.

Example 1. Consider the case $e = 415 = (110011111)_2$. Then, the binary addition chain with length $l = 14$ for that e is,

$$U := 1 \rightarrow 2 \rightarrow 4 \rightarrow 8 \rightarrow 16 \rightarrow 32 \rightarrow 64 \rightarrow 128 \rightarrow$$
$$256 \rightarrow 384 \rightarrow 400 \rightarrow 408 \rightarrow 412 \rightarrow 414 \rightarrow 415$$

With the associated sequence governed by the rule, $u_i = u_{i-1} + u_{i-1} = 2u_{i-1}$ for $1 \leq i \leq 8$ and, $u_9 = u_8 + u_7$, and $u_{10+m} = u_{9+m} + u_{4-m}$, for $m = 0, 1, \ldots, 4$.

2.2 Artificial Immune System and Problem Representation

Our algorithm is based on a mechanism called clonal selection principle [10], that explains the way on which the antibodies eliminate a foreign antigen.

Fig. 1 depicts the clonal selection principle, that establishes the idea that only those antibodies that best match the antigen are stimulated. These stimulated antibodies are reproduced by cloning and the new clones suffer a mutation process with high rates (called hypermutation). After this process is done some of the newly created antibodies will increase their affinity to the antigens. Those clones will neutralize and eliminate the antigens. Once that the foreign antigens have been exterminated, the immune system must return to its normal values, eliminating the exceeding antibodies cells. However, the best cells remain into the body as memory cells. According to de Castro and Timmis [11] in every

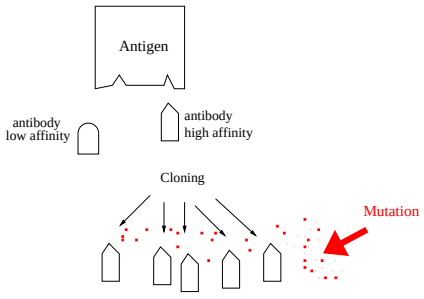

Fig. 1. The Clonal Selection Principle of the Immune System

artificial immune system, as in any other computational system with biological inspiration, the following elements must be defined:

A representation of the system components: A foreign antigen will be represented as the exponent e that we wish to reach. Antibodies will be represented by the addition chain sequence that contains the arithmetic recipe that we need to compute so that we can achieve the desired goal (the antigen). For instance, if we wish to reach the antigen $e = 415$ we can execute the antibody addition chain sequence: $1 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow 10 \rightarrow 20 \rightarrow 40 \rightarrow 80 \rightarrow 90 \rightarrow 180 \rightarrow 360 \rightarrow 400 \rightarrow 410 \rightarrow 415$ which is a feasible problem solution (although not necessarily the best as we will see below).
Mechanisms to evaluate the interaction of individuals with the environment and each other: The fitness of a given antibody is determined by the length of its corresponding addition chain. The shorter the length is the better the associated fitness.
Procedures of adaptation that govern the dynamics of the system: The dynamics of our system is based on the clonal selection principle.

Input: An exponent (antigen) e.
Output: An optimal addition chain (antibody) U.
Procedure AIS_Optimal_Addtion_Chain(e, U)
1. Start with an initial population consisting of three antibodies Ab_1, Ab_2 and Ab_3 constructed according to the following rules,
   1.a If the antigen e is an even number of the form $e = 2^k e'$ then save k and work with $e = e'$, with e' an odd number.
   1.b Pre-initialize the antibody Ab_1 using the sequence: **1 - 2 3**
   1.c Pre-initialize the antibody Ab_2 using the sequence: **1 - 2 4**
   1.d Pre-initialize the antibody Ab_3 using the sequence: **1 - 2 - 3 5**
   1.e Complete the addition chain sequence for the three germinal antibodies Ab_1, Ab_2 and Ab_3 needed to achieve the antigen $e = e'$ trying to use the doubling rule, $u_i = u_{i-1} + u_{i-1} = 2u_{i-1}$ as much as possible.
   1.f Add to each antibody the k doubling remaining steps needed to reach the original antigen e of step 1.*a*. Now we have three valid addition chains associated to each one of the antibodies Ab_1, Ab_2 and Ab_3, that allow us to achieve the desired antigen e.
2. Repeat:
   3. Compute the associated fitness values of the three constructed antibodies, i.e., the corresponding addition sequence lengths. Assign a better fitness to the antibodies with shorter lengths.
   4. Determine how many clones (copies) will be made per each antibody. The better the fitness the more clones to be made.
   5. **Antibody Cloning**: Apply the mutation operator to each clone as follows,
   5.a Randomly select a mutation point i and a random number j such that $0 \leq j < i < e$
   5.b The new value of the clone's addition chain at the mutation point u_i will be $u_i = u_{i-1} + u_j$
   5.c Update the upper part of the clone's addition chain so that the antigen e is still reached by Applying as much doubling steps as possible.
   6. From the set of original antibodies and modified clones, select the top three and discard the rest. Those three survivor antibodies will make it to the next generation.
7. Go to step 2 a predetermined number of generations.

Fig. 2. Finding Optimal Addition Chains Using an AIS Strategy

2.3 AIS Algorithm

The AIS strategy to compute optimal addition chains is shown in Fig. 2. The exponent e is named the antigen or goal that the artificial immune system is trying to achieve. Starting with an initial population of three antibodies, the algorithm of Fig. 2 uses the cloning mechanism to generate slightly different replicas that are then selected based on the fitness of the individuals. A clone fitness is measured in terms of the length of its corresponding addition chain. In order to illustrate how our algorithm computes its task, let us consider the case when we want to obtain an optimal addition chain for the antigen $e = 415$.

Example 2. Given the antigen $e = 415$, then the algorithm of Fig. 2 performs as follows,

1. It starts with an initial population consisting of three antibodies Ab_1, Ab_2 and Ab_3 constructed according to the following rules,
   (a) Since the antigen e is an odd number, it proceeds directly to construct the three original antibodies.
   (b) Construction of the initial antibody Ab_1 using: 1 - 2 - 3
   (c) Construction of the init2ial antibody Ab_2 using: 1 - 2 - 4
   (d) Construction of the initial antibody Ab_3 using: 1 - 2 - 3 - 5
   (e) It proceeds to complete the antibodies. After performing as many doubling steps as possible we get,
   Ab_1 : **1-2-3**-6-12-24-48-96-192-384-408-414-415
   Ab_2 : **1-2-4**-8-16-32-64-128-256-384-400-408-412-414-415
   Ab_3 : **1-2-3-5**-10-20-40-80-160-320-400-410-415
2. Repeat
3. Therefore, the corresponding fitness values (addition chain lengths) for each antibody are: $Ab_1 = 12$, $Ab_2 = 14$, $Ab_3 = 12$.
4. Since the fittest antibodies are Ab_1 and Ab_2, four clones of each one of them are made but only one clone of Ab_3.
5. **Antibody Cloning**: As an illustrative example let us analyze a clone Cl which is an exact replica of Ab_3 given as,
   Cl : 1-2-3-5-10-20-40-80-160-320-400-410-415.
   For this case the mutation mechanism will be applied as follows:
   (a) A mutation point i and a random number j, say 160 and 10, respectively, are randomly selected.
   (b) The number 160 will be changed by 90, which is obtained from the addition of 80+10. Therefore, the mutated (and mutilated) clone will look as, 1-2-3-5-10-20-40-80-**90**.
   (c) The mutilated clone Cl must be restored by using as many doubling steps as possible until the antigen e is reached. Hence:
   Cl:**1-2-3-5-10-20-40-80-90**-180-360-400-410-415
6. From the whole population, the algorithm selects the best three solutions. They will be the antibodies for the next generation.
7. Go to step 2 a predetermined number of generations.

As it was explained, the above procedure creates nine clones at each iteration. After 100 generations our algorithm was able to find the following addition chain of length $l = 11$,

$$U : 1 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow 10 \rightarrow 20 \rightarrow 40 \rightarrow 80 \rightarrow 83 \rightarrow 166 \rightarrow 332 \rightarrow 415 \quad (1)$$

The result obtained means a saving of 3 operations with respect to the binary method that, as it was seen in example 1, yields an addition chain of 14.

Proposition 1. *The addition chain U of Eq. 1 is the shortest addition chain for $e = 415$.*

3 Multiplicative Inversion over GF(2^m)

Since a quite long time it has been known that Fermat's Little Theorem (FLT) provides a mechanism to compute multiplicative inverses in finite fields [1]. Certainly, FLT establishes that for any nonzero element $\alpha \in GF(2^m)$, the identity $\alpha^{-1} \equiv \alpha^{2^m-2}$ holds. Therefore, multiplicative inversion can be performed by computing,

$$\alpha^{2^m-2} = \alpha^{2^1} \times \alpha^{2^1} \times \cdots \alpha^{2^{m-1}} \quad (2)$$

A straightforward implementation of Eq. 2 can be carried out using the binary exponentiation method, which requires $m-1$ field squarings and $m-2$ field multiplications. The Itoh-Tsujii Multiplicative Inverse Algorithm (ITMIA) [8] on the other hand, reduces the required number of multiplications to $k+hw(m-1)-2$, where $k = \lfloor log_2(m-1) \rfloor$ and $hw(m-1)$ are the number of bits and the Hamming weight of the binary representation of $m-1$, respectively. This remarkably saving on the number of multiplications is based on the observation that since $2^m - 2 = (2^{m-1}-1) \cdot 2$, then the identity from Fermat's little theorem can be rewritten as $\alpha^{-1} \equiv \alpha^{2^m-2} \equiv \alpha^{(2^{m-1}-1)^2}$. Then ITMIA computes the field element $(2^{m-1}-1)$ using a recursive re-arrangement of the finite field operations. To see how this can be carried out let us first make some definitions.

Definition 1. *Let α be any arbitrary nonzero element in the field $GF(2^m)$. Then we define $\beta_k \in GF(2^m), k$ a positive integer, as*

$$\beta_k = \alpha^{2^k-1} \quad (3)$$

Lemma 1. *Let k, j be two positive integers. Then, the element $\beta_{k+j} \in GF(2^m)$ can be expressed as,*

$$\beta_{k+j} = {\beta_k}^{2^j} \cdot \beta_j = {\beta_j}^{2^k} \cdot \beta_k \quad (4)$$

Proof. Using Definition 1 to substitute β_k and β_j in terms of the variable α we obtain,

$$\begin{aligned}\beta_k{}^{2^j} \cdot \beta_j &= (\alpha^{2^k-1})^{2^j} \cdot \alpha^{2^j-1} \\ &= \alpha^{2^{k+j}-2^j} \cdot \alpha^{2^j-1} \\ &= \alpha^{2^{k+j}-2^j+2^j-1} \\ &= \alpha^{2^{k+j}-1} \;=\; \beta_{k+j}\end{aligned}$$

□

Theorem 1 (Itoh-Tsujii Algorithm). *Let α be any arbitrary nonzero element in the field $GF(2^m)$. Let us consider the binary expansion of the k-bit positive number $m-1$ and let us assume that the nonzero components of that binary expansion can be listed as,*

$$m-1 = 2^{l_1} + 2^{l_2} + \cdots + 2^{l_{t-1}} + 2^{l_t}.$$

Where $l_1 < l_2 < \cdots < l_{t-1} < l_t = k-1$.

Then the multiplicative inverse of α, namely $\alpha^{-1} \in GF(2^m)$, can be written as,

$$\begin{aligned}\alpha^{-1} = \alpha^{2^m-2} \;&=\; \left(\alpha^{2^{m-1}-1}\right)^2 \\ &= \left[\beta_{2^{l_1}}\left(\beta_{2^{l_2}}\left(\cdots\beta_{2^{l_{t-2}}}\left[\beta_{2^{l_{t-1}}}\left(\beta_{2^{l_t}}\right)^{2^{2^{l_{t-1}}}}\right]^{2^{2^{l_{t-2}}}}\cdots\right)^{2^{2^{l_2}}}\right)^{2^{2^{l_1}}}\right]^2 \qquad (5)\end{aligned}$$

Moreover, the overall complexity of Eq. 5 is $m-1$ field squarings plus $N = k + hw(m-1) - 2$ field multiplications, where $k = \lfloor log_2(m-1) \rfloor$ and $hw(m-1)$ are the number of bits and the Hamming weight of the binary representation of $m-1$, respectively

Proof. See [8, 9].

3.1 A Design Example

The concept of addition chains leads us to a natural way to generalize the Itoh-Tsujii Algorithm. Consider the algorithm shown in Fig. 3. That algorithm iteratively computes the β_i coefficients in the exact order stipulated by the addition chain U. Indeed, starting from $\beta_0 = (\alpha)^{2^{u_0}-1} = (\alpha)^{2^{u_0}-1} = \alpha^{2^1-1}$, the algorithm computes the other l β_i coefficients. In the final iteration, after having computed the coefficient $\beta_l = (\alpha)^{2^{m-1}-1}$, the algorithm returns the required multiplicative inversion by performing a regular field squaring, namely, $\beta_l^2 = (\alpha^{2^m-2}) = \alpha^{-1}$.

Input: An element $\alpha \in GF(2^m)$, an addition chain U of length l for $m-1$ and its associated sequence V.
Output: $\alpha^{-1} \in GF(2^m)$
Procedure MultiplicativeInversion(α, U)
1. $\beta_0 = \alpha$
2. for i from 1 to l do:
3. $\quad \beta_i = (\beta_{i_1})^{2^{u_{i_2}}} \cdot \beta_{i_2}$
4. return $(\beta_t{}^2)$;

Fig. 3. An Algorithm for Multiplicative Inversion using a Generalized Itoh-Tsujii Approach

Table 1. Algorithm of Fig. 3: β_i Coefficient Generation

i	u_i	rule	$\beta_{i_1}^{2^{u_{i_2}}} \cdot \beta_{i_2}$	$\beta_i = \alpha^{2_i^u - 1}$
0	1	–	–	$\beta_0 = \alpha^{2^1-1}$
1	2	$2u_{i-1}$	$\beta_0^{2^1} \cdot \beta_0$	$\beta_1 = \alpha^{2^2-1}$
2	3	$u_{i-1} + u_{i-2}$	$\beta_1^{2^1} \cdot \beta_0$	$\beta_2 = \alpha^{2^3-1}$
3	5	$u_{i-1} + u_{i-2}$	$\beta_2^{2^2} \cdot \beta_1$	$\beta_3 = \alpha^{2^5-1}$
4	10	$2u_{i-1}$	$\beta_3^{2^5} \cdot \beta_3$	$\beta_4 = \alpha^{2^{10}-1}$
5	20	$2u_{i-1}$	$\beta_4^{2^{10}} \cdot \beta_4$	$\beta_5 = \alpha^{2^{20}-1}$
6	40	$2u_{i-1}$	$\beta_5^{2^{20}} \cdot \beta_5$	$\beta_6 = \alpha^{2^{40}-1}$
7	80	$2u_{i-1}$	$\beta_6^{2^{40}} \cdot \beta_6$	$\beta_7 = \alpha^{2^{80}-1}$
8	83	$u_{i-1} + u_2$	$\beta_7^{2^3} \cdot \beta_2$	$\beta_8 = \alpha^{2^{83}-1}$
9	166	$2u_{i-1}$	$\beta_8^{2^{83}} \cdot \beta_8$	$\beta_9 = \alpha^{2^{166}-1}$
10	332	$2u_{i-1}$	$\beta_9^{2^{166}} \cdot \beta_9$	$\beta_{10} = \alpha^{2^{332}-1}$
11	415	$u_{i-1} + u_{i-3}$	$\beta_{10}^{2^{166}} \cdot \beta_9$	$\beta_{11} = \alpha^{2^{415}-1}$

Example 3. Let $\alpha \in GF(2^{415})$ be an arbitrary nonzero field element. Then, using the addition chain of Example 2, the algorithm of Fig. 3 will compute the sequence of β coefficients as shown in Table 3.1. Once again, notice that after having computed the coefficient β_{11} the only remaining step is to obtain α^{-1} as, $\alpha^{-1} = \beta_{11}^2$

4 Result Comparison

We compare the results obtained by our algorithm against the modified factor method presented by Takagi et al [9] (whose results are the best known to date) and the ITMIA binary method [8]. Table 2 shows the optimal addition chains for m=32k (which is an important exponent form for error-correcting code applications). The chains found by the AIS algorithm are summarized in the second and third columns. On a total of seven cases the AIS algorithm presented here outperforms the method of [9]. And in all the cases, those two algorithms outperform the ITMIA binary method.

m	$m-1$	AIS	AIS	Takagi [9]	ITMIA Binary method [8]
32	31	1 2 3 6 7 14 28 31	7	7	8
64	63	1 2 3 6 7 14 28 56 63	8	8	10
96	95	1 2 3 5 10 20 40 80 90 95	9	9	11
128	127	1 2 3 6 12 24 48 96 120 126 127	10	10	12
160	159	1 2 3 6 12 24 48 96 144 156 159	10	10	12
192	191	1 2 4 8 16 17 34 68 136 170 187 191	11	11	13
224	223	1 2 3 6 12 13 26 52 104 208 221 223	11	11	13
256	255	1-2-3-5-10-20-40-80-85-170-255-	10	10	14
288	287	1-2-3-5-7-14-28-56-112-224-280-287-	11	11	13
320	319	1-2-3-6-12-18-36-72-144-288-306-318-319-	12	12	14
352	351	1-2-3-6-12-24-27-54-108-216-324-351-	11	11	14
384	383	1-2-3-5-10-20-40-80-160-320-360-380-383-	12	13	15
416	415	1-2-3-5-10-20-40-80-83-166-332-415-	11	12	14
448	447	1-2-3-6-12-18-36-72-144-288-432-444-447-	12	12	15
480	479	1-2-3-6-7-14-28-56-112-224-448-476-479-	12	13	15
512	511	1-2-3-5-10-15-30-60-120-240-480-510-511-	12	12	16
576	575	1-2-3-5-10-20-23-46-92-184-368-552-575-	12	13	15
608	607	1-2-3-6-12-18-36-72-144-288-576-594-606-607-	13	13	15
640	639	1-2-3-6-12-24-26-52-104-208-416-624-636-639-	13	13	16
704	703	1-2-3-5-10-20-40-80-160-320-640-680-700-703-	13	13	16
736	735	1-2-3-5-10-15-30-60-120-240-480-720-735-	12	12	16
768	767	1-2-3-5-10-20-40-80-83-166-332-664-747-767-	13	14	17
800	799	1-2-3-6-12-24-48-96-192-384-768-792-798-799-	13	13	15
832	831	1-2-3-6-12-24-48-96-192-384-768-816-828-831-	13	13	16
864	863	1-2-3-5-10-20-40-43-86-172-344-688-860-863-	13	15	16
896	895	1-2-3-5-10-20-40-80-160-163-326-652-815-895-	13	14	17

Table 2. Optimal addition chains for $m = 32k$. AIS=Artificial Immune System

p	$p-1$	AIS	AIS	ITMIA Binary method [8]
163	162	1 2 4 8 16 32 64 80 81 162	9	9
167	166	1 2 3 5 10 20 40 80 83 166	9	10
173	172	1 2 3 5 10 20 40 43 86 172	9	10
191	190	1 2 3 5 10 20 40 80 160 180 190	10	12
193	192	1 2 3 6 12 24 48 96 192	8	8
197	196	1 2 4 8 16 32 48 49 98 196	9	9
223	222	1 2 3 6 12 24 48 96 192 216 222	10	12
2 233	232	1 2 4 8 16 24 28 29 58 116 232	10	10
269	268	1 2 4 8 16 32 64 66 67 134 268	10	10
271	270	1 2 3 5 10 20 40 80 90 180 270	10	11
293	292	1 2 4 8 16 32 64 72 73 146 292	10	10
331	330	1 2 3 5 10 20 40 80 160 320 330	10	11
379	378	1 2 4 8 16 18 36 72 144 288 360 378	11	13
383	382	1 2 3 5 10 20 40 80 160 320 360 380 382	12	14
389	388	1 2 4 8 16 32 64 96 97 194 388	10	10
443	442	1 2 3 6 12 13 26 52 104 208 416 442	11	13
463	462	1 2 4 8 16 32 33 66 132 264 396 462	11	13
491	490	1 2 3 5 10 20 40 80 160 320 480 490	11	13
509	508	1 2 3 6 12 14 28 30 60 120 240 480 508	12	14
521	520	1 2 4 8 16 32 64 65 130 260 520	10	10

Table 3. Optimal addition chains for $e = p-1$, p a prime.

Table 3 summarizes the results obtained by AIS and the binary method for $e = p$ number, with p a prime number (which is an important exponent form for elliptic curve cryptography). In most of the cases, the immune algorithm obtains better results than the ITMIA binary method.

5 Conclusions

In this paper we described how an artificial immune system can be applied to the problem of finding optimal addition chains for field exponentiation computations. The results obtained by our scheme yield the shortest reported lengths for exponents typically used when computing field multiplicative inverses for error-correcting and elliptic curve cryptographic applications. Future work includes finding addition chains for bigger exponents such as the ones typically used in RSA and DSA cryptosystems. We would like also to explore the feasibility of applying other biological-inspired heuristics to the optimal addition chain problem.

References

1. Knuth, D.E.: The Art of Computer Programming 3rd. ed. Addison-Wesley, Reading, Massachusetts (1997)
2. Menezes, A.J., van Oorschot, P.C., A.Vanstone, S.: Handbook of Applied Cryptography. CRC Press, Boca Raton, Florida (1996)
3. Gordon, D.M.: A survey of fast exponentiation methods. Journal of Algorithms **27** (1998) 129–146
4. Lamont, G.B., Marmelstein, R.E., Veldhuizen, D.A.V.: A distributed architechture for a self-adaptive computer virus immune system. In: New Ideas in Optimization. Mc Graw-Hill (1999) 167–183
5. Forrest, S., Hofmeyr, S.A.: Immunology as Information Processing. In Segel, L., Cohen, I., eds.: Design Principles for the Immune System and Other Distributed Autonomous Systems. Santa Fe Institute Studies in the Sciences of Complexity. Oxford University Press (2000) 361–387
6. Coello Coello, C.A., Cruz-Cortés, N.: A parallel implementation of an artificial immune system to handle constraints in genetic algorithms: Preliminary results. In: Proceedings of the special sessions on artificial immune systems in the 2002 Congress on Evolutionary Computation, 2002 IEEE World Congress on Computational Intelligence, Honolulu, Hawaii (2002)
7. Toma, N., Endo, S., Yamada, K.: Immune Algorithm with Immune Network and MHC for Adaptive Problem Solving. In: Proceedings of the IEEE System, Man, and Cybernetics. Volume IV. (1999) 271–276
8. Itoh, T., Tsujii, S.: A fast algorithm for computing multiplicative inverses in GF(2^m) using normal basis. Information and Computing **78** (1988) 171–177
9. Takagi, N., Yoshiki, J., Tagaki, K.: A fast algorithm for multiplicative inversion in GF(2^m) using normal basis. IEEE Transactions on Computers **50(5)** (2001) 394–398
10. Burnet, F.M.: Clonal selection and after. In Bell, G.I., Perelson, A.S., Jr., G.H.P., eds.: Theoretical Immunology, Marcel Dekker Inc. (1978) 63–85
11. de Castro, L., Timmis, J.: An Introduction to Artificial Immune Systems: A New Computational Intelligence Paradigm. Springer-Verlag (2002)