Genetic Programming for Improved Cryptanalysis of Elliptic Curve Cryptosystems

Tim Ribaric
Department of Computer Science
Brock University
St. Catharines ON
Email: tribaric@brocku.ca

Sheridan Houghten
Department of Computer Science
Brock University
St. Catharines ON
Email: shoughten@brocku.ca

***Abstract*—Public-key cryptography is a fundamental component of modern electronic communication that can be constructed with many different mathematical processes. Presently, cryptosystems based on elliptic curves are becoming popular due to strong cryptographic strength per small key size. At the heart of these schemes is the intractability of the elliptic curve discrete logarithm problem (ECDLP).**

Pollard's Rho algorithm is a well known method for solving the ECDLP and thereby breaking ciphers based on elliptic curves. It has the same time complexity as other known methods but is advantageous due to smaller memory requirements. This paper considers how to speed up the Rho process by modifying a key component: the iterating function, which is the part of the algorithm responsible for determining what point is considered next when looking for a collision. It is replaced with an alternative that is found through an evolutionary process. This alternative consistently and significantly decreases the number of iterations required by Pollard's Rho Algorithm to successfully find a solution to the ECDLP.

I. Introduction

Public key cryptography is the central component of present day secured communication. This began with the Diffie-Hellman key exchange [1], followed shortly thereafter by the Rivest, Shamir, and Adleman (RSA) method [2] which is actively used in present day internet communication. The basis of these schemes is that an asynchronous key is created that has two components: a private key and a public key. A message is encoded with the public key and transmitted to the other party. The other party uses the corresponding private key to decode the message. The system is constructed in such a way that the public key can be shared to all while it remains computationally infeasible to determine the private key. Different cryptographic schemes use different mathematical operations to ensure that the system is reasonably secure. With RSA, the difficult computational process is factorization of integers that are the product of large prime numbers.

In elliptic curve cryptography this computational difficulty is accomplished through the Elliptic Curve Discrete Logarithm Problem (ECDLP). We represent a point Q on an elliptic curve restricted to a field with the following notation: $Q = (Q.x, Q.y)$, where the point Q has x and y coordinates. Let k represent some scalar. Then the cryptosystem is arranged as follows: $Q = kP$, where point Q can represent the public key of the system and point P can represent the private key. It is known to be computationally difficult to determine the value of k if only kP is provided. Finding an effective method with a reasonable running time to solve this would effectively crack the cryptosystem. One of the more popular forms of solving the ECDLP is Pollard's Rho Algorithm. With the Rho process we seek to find the value of k by determining two equations involving points in the field that equal one another when multiplied by different scalar values. With this presented notation we are attempting to find the solution to: $c^{'}P + d^{'}Q = c^{''}P + d^{''}Q$. More precisely, we attempt to find the scalar values of c',c'',d', and d''. Once these values are known we can find the value of k by using a field inversion operation and evaluating $k = (c^{'} - c^{''})(d^{''} - d^{'})^{-1}\ mod\ n$, where n is a large prime.

This study utilizes a genetic programming approach to aid in the calculation of the ECDLP. Specifically, the number of iterations required by the Rho process are significantly reduced by using an evolved genetic program in place of the usual iterating function that is a central component of the Rho Algorithm.

Other computational intelligence techniques have been applied in various aspects of cryptography. For example, genetic algorithms have been employed in generating keys for elliptic curve cryptosystems [3]. With respect to cryptanalysis, genetic algorithms have been used in various ways to attack classical ciphers, block ciphers and stream ciphers [4] [5] [6] [7], and to analyze weak keys for ciphers [8]. Laskari et al [9] provide a comprehensive overview of the application of computational intelligence techniques used in cryptographic and cryptanalysis studies. Alongside this survey, original research is presented on a number of different experiments that treat cryptography problems as discrete optimization tasks. Among those presented is an experiment that considers the use of an artificial neural network to find the least significant bit of an ECDLP solution. Finding this value leads to a reduction in the computation time required to calculate the rest of the solution. Field sizes of 14, 20, and 32 bits were investigated. This technique managed to identify the correct solution after training at an average rate of 57%.

The method used in the current study is to employ genetic programming to reduce ECDLP calculation time of an established method (Pollard's Rho Algorithm) that effectively has a

978-1-5090-4601-0/17/$31.00 ©2017 IEEE

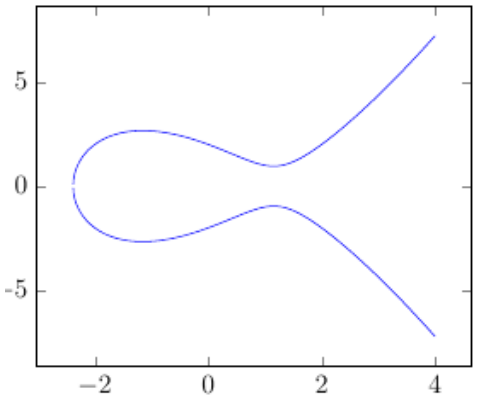

Fig. 1. $y^2 = x^3 - 4x + 4$

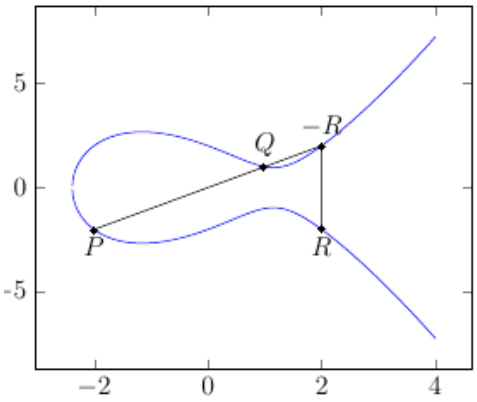

Fig. 2. Demonstration of Point Addition

success rate of 100%, as explained in the following section. To the best of our knowledge there have been no similar studies to the one presented in this paper.

II. Background

A. Elliptic Curve Discrete Logarithm Problem (ECDLP) and Cryptography

The use of Elliptic Curve Cryptography was first proposed independently by Koblitz [10] and Miller [11]. These proposed systems made use of an elliptic curve E that is defined as the set of solutions (x, y) where $a, b \in \mathbb{Z}$ to the following equation:

$$y^2 = x^3 + ax + b \tag{1}$$

The values a, b must be chosen so that $4a^3 + 27b^2 \neq 0$. If this requirement is not met then the resulting curve will be unsuitable for cryptographic use [12]. Additionally if the curve used fulfills this criteria then the Rho Algorithm, described in the following section, will always successfully find the solution to the ECDLP [10]. Fig. 1 shows an example curve plotted on a Cartesian plane.

The curve is symmetric about the x-axis. Due to this symmetry many properties arise. For instance any line between two distinct points on the curve will intersect the curve at most once. It is possible to define an *addition* operation to add two points on the curve. In all cases the first step is to draw a line between the two points. In total there are four different scenarios:

- **Points are both at the origin**: A tangent is drawn to the origin and is said to extend to infinity.
- **Points share the same x-coordinate**: A line is drawn between the two points and it is said to extend to infinity.
- **Line between the points does not intersect the curve**: In this case as well the line is said to extend to infinity.
- **Line between the points intersects the curve**: Here the addition is calculated using the *chord and tangent* method. If P, Q are points on E and $P \neq Q$ (i.e. not any of the previously defined scenarios) then one simply draws a line between the two points and it will intersect the curve at a third point $-R$, which is reflected in the x-axis to produce R. This is demonstrated in Fig. 2.

In order to complete this definition so that it constitutes a group, an identity element $\mathcal{O}$ called the *point at infinity* is introduced. This element satisfies $P + \mathcal{O} = P$ for every P. What this means is that adding any two points on the curve will always result in a point on the curve or the point at infinity. With this point the group is closed under point addition. In summary elliptic curves form a finite Abelian group when defined over $\mathbb{F}_n$, where n is a large prime number.

Another operation, *point negation*, is easy to calculate. If $P = (P.x, P.y)$ then $-P$ is defined as $(P.x, -P.y)$.

It is possible to define the addition operation explicitly so that it can be calculated without explicitly knowing all of the points on the curve. If $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ are points on E neither of which is $\mathcal{O}$ then $P + Q = (x_3, y_3)$ where:

$$x_3 = \lambda^2 - x_1 - x_2$$
$$y_3 = \lambda(x_1 - x_3) - y_1$$

If $P = Q$ then

$$\lambda = \frac{3x_1^2 + a}{2y_1} \tag{2}$$

If $P \neq Q$ then

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1} \tag{3}$$

It is possible to define a multiplication operation over points as well. In fact this operation can be seen as multiple applications of point arithmetic defined above, for example $2P = P + P$. This operation is often referred to as *point doubling* when the scalar value is 2, while in all other cases this operation is referred to as *point multiplication*.

Let E be an Elliptic Curve defined over $\mathbb{F}_n$, where n is a large prime. Let $P \in E$ be a point of prime order q. Let $\langle P \rangle$ be the prime order subgroup of E generated by P. If $Q \in \langle P \rangle$ then $Q = kP$ for some scalar k where $0 \leq k \leq q$. The problem is then finding k given P, Q, and the parameters

of curve E. The ECDLP can be thought of as the elliptic curve variant to the more common logarithm problem. For example with $y = n^x$, it is easy to calculate y given n and x. However when given y and n it is difficult to calculate x. When calculating this logarithm the result is often a real value. When calculating the logarithm over a field, as in the case of the ECDLP, the resulting value is a member of the field, or in other words it is composed of an x and y coordinate that are both integers. Calculating a logarithm in a field is often referred to as the *discrete logarithm problem*.

B. Pollard's Rho Algorithm

The original method, proposed by Pollard [13], was used to find the prime roots of a composite number, and was constructed in such a way that it could be coded into a programmable calculator. Soon after a slightly revised method was proposed by Pollard [14] that was meant to compute the index of any integer to a given primitive root of a prime p. The additional benefit of the Rho algorithm is that it has the best run-time of any known method of solving the discrete logarithm problem. Similarly the memory requirements are also negligible, making the algorithm very appealing to use.

C. Pollard's Rho Algorithm for Elliptic Curves

Algorithm 1 Pollard's Rho Algorithm for Elliptic Curves [12]

Input: $P \in E(\mathbb{F}_q)$ of prime order n, $Q \in \langle P \rangle$
Input: Partition function $EvoH : \langle P \rangle \rightarrow \{1, 2, ..., L\}$
1: **for** j from 1 to L **do**
2: Select $a_j, b_j \in_R [0, n-1]$
3: Compute $R_j = a_j P + b_j Q$
4: **end for**
5: Select $c', d' \in_R [0, n-1]$
6: Compute $X' = c'P + d'Q$
7: $X'' \leftarrow X'$, $c'' \leftarrow c'$, $d'' \leftarrow d'$
8: **repeat**
9: $j = EvoH(X')$
10: $X' \leftarrow X' + R_j$
11: $c' \leftarrow c' + a_j \ mod \ n$
12: $d' \leftarrow d' + b_j \ mod \ n$
13: **for** i from 1 to 2 **do**
14: $j = EvoH(X'')$
15: $X'' \leftarrow X'' + R_j$
16: $c'' \leftarrow c'' + a_j \ mod \ n$
17: $d'' \leftarrow d'' + b_j \ mod \ n$
18: **end for**
19: **until** $X' = X''$
20: **if** $d' = d''$ **then**
21: **return** False
22: **else**
23: $k = (c' - c'')(d'' - d')^{-1} \ mod \ n$
24: **return** k
25: **end if**

The method used for this experiment is a modified version of the Rho presented in Hankerson et al. [12] and seen in Algorithm 1. The inputs supplied to the algorithm are two points P, Q. The point Q is expressed as kP, some scalar multiple of point P. The value of k is the solution to our discrete logarithm problem. Next, we require a hash function that will allow us to map any random point to a value between 1 and L, where L is the number of sections into which the curve will be divided while we are searching for the value of k. Pollard's original formulation of the Rho algorithm used 3 sections. In most constructions of the Rho algorithm the hash function is simply implemented with $(P.x) \ mod \ L + 1$.

The purpose of the current study is to instead use genetic programming to determine a more effective hash function $EvoH$. This study fixed the value of L, i.e. the number of sections, at 32. This value was selected as it is a common choice for implementations of the Rho Algorithm [12].

The algorithm begins by filling a random buffer R of L values in the range 1 to L. These values are pairwise multiplied with L random points in the curve. Next two random values c' and d' are chosen from the interval. Then X' is calculated, which is comprised of the sum of point multiplications of $c'P$ and $d'Q$. For the first iteration $c^{'}$ is set to the same value as $c^{''}$ and similarly $d^{'}$ and $d^{''}$ are set to the same value. The Rho algorithm then proceeds by calculating two different sequences of points; this process is more commonly known as Floyd's cycle detecting algorithm [15]. When these two sequences yield the same point (line 19), the process then attempts to find the value of k (line 23) which, as described earlier, will always be successful due to the construction of the ECDLP problem. What is worth noting is that the $EvoH$ hash function guides the choice of what intermediate point is considered next.

The current study examines whether making the hash function distribute values more randomly will reduce the runtime of the Rho Algorithm, i.e. to find the value of k in fewer iterations. Specifically, an evolved genetic program will be used to replace the original hash function.

The parameters selected by the algorithm are a_j, b_j, c', and d'. The first two of these parameters are randomly selected series of values in the range of 0 to $n-1$, as found in line 2 of Algorithm 1. The second two are scalar parameters that were statically assigned to be evenly spaced in field size n, specifically $c' = n/4$ and $d' = 3n/4$ using integer division (line 5). Because of how these parameters are assigned, when looking at the same values of P and Q, subsequent runs of the algorithm differ only if the values of a_j and b_j are varied. Those two series are the only independent variables in the Rho algorithm.

D. Research on Pollard's Rho Algorithm

Since the first publication of Pollard's Rho Algorithm there has been substantive research into improving the method. The refinement that has resulted in the best improvement of runtime has been the introduction of a parallel version of the algorithm [16]. In this case there is a speed up factor of *M*, where *M* is the number of processors assigned to calculating intermediate values. There have been other refinements to the

process that have also produced a reduction in runtime. The use of $R + S$ additive walks [17] modified the composition of the iterating hash function. Instead of using the typical 3 subsets outlined in the original method by Pollard [14] a varying amount of subsets are generated. Further offsets into those subsets are also employed in an attempt to further scatter the iterated points throughout the subset. In general it was found that the higher the number of subsets utilized the better the performance of the method.

An additional refinement uses negation maps [18] to partition the points in the subsections of the iterating function. Instead of using point addition as in the original method, both P and $-P$ are calculated. The point with the smaller y coordinate is used in the iterating function, as there is a decrease in the computations that need to be performed. However it is possible for calculations to get trapped in never-ending loops using this scheme requiring a method for loop detection and escape to be implemented.

The same research team also developed a method referred to as *point-halving* [19]. In this case the iterating function is modified, with point-halving used instead of point addition since point-halving is computationally easier. Similarly a method of *point subtraction* [20] has also been developed that modifies the iterating function to short cut the complexity of the calculations by subtracting points in the field instead of adding them together. The same research team also proposed a method [21] of decreasing the amount of time needed to perform collision detection, however this increase in efficiency does not surpass the gains seen when using the parallel formulation of the process. Lastly the same research team discovered another modification, when restricted to finite extension fields [22] a decrease in calculation complexity can be seen by utilizing the fact that exponential point calculations are just cyclic shifts of the underlying fields.

III. Proposed Method

The purpose of this study is to employ a genetic programming technique to decrease the number of iterations performed by the Rho Algorithm, thus finding the solution to the ECDLP in less computation time.

A. Genetic Programming

Genetic Programming is a metaheuristic popularized by Koza [23] which mimics natural evolution. The goal of this study is to evolve a highly fit expression tree to represent the iterating hash function by first creating an initial population of randomly generated expression trees based on simple components referred to as *terminal nodes* and *internal nodes* and then evolving this population through successive generations with the goal of finding the most fit final expression tree. Fitness is determined via the application of a fitness function that will assign a numerical score to each expression tree considered. With each generation this new population is created by probabilistically applying reproduction and mutation and performing a selection process that picks the most fit individuals to survive to the next generation.

TABLE I
Runtime Parameters

Parameter	Value
Population Size	1000
Generations	100
Max Depth of Program Trees	17
Crossover Rate	0.9
Mutation Rate	0.1
Tournament Size	5

TABLE II
Node values

Terminal Node	Description
$P.y$	Y coordinate of point
Internal Node	**Description**
Addition	Integer sum of two operands
Subtraction	Integer difference of the two operands
Multiplication	Integer product of two operands
Negation	Negation of a single operand

In this scheme reproduction is performed by a single point cross-over technique. Two expression trees selected via probabilistic methods randomly pick a node. The two trees are split at this node and swap the resulting sub-trees. Mutation is performed by probabilistically selecting an expression tree and probabilistically regenerating a simple sub-tree at a random node. Selection was performed by tournament selection. Additional runtime parameters are summarized in Table I and were determined empirically. This study was conducted using an evolutionary algorithm software modelling package called Distributed Evolutionary Algorithms in Python (DEAP) [24].

B. Terminals and Internal Nodes

The expression tree generated by the evolutionary process is comprised of two components: *terminal*, or *leaf*, nodes and *internal* nodes. These components are summarized in Table II and were determined via empirical analysis. Fig. 3 shows an example expression tree formed during the course of a run.

C. Fitness Function

The fitness function is an integral part of the genetic programming process. The rationale for the choice of function here is to randomize the values produced by the $EvoH$ hash function. The supposition is that if it is adequately random the next points considered by the Rho process will be well distributed through the curve and result in a lower number of iterations to find the value of k. Since each curve is unique, as well as the choices of P and Q, the iterating function used for each Rho process should be created in a way that maximizes randomness for the choices of parameters.

```
operator.sub(operator.neg(P.y),
operator.mul(operator.mul(P.y, P.y),
operator.add(P.y, P.y)))
```

Fig. 3. Example of Evolved Partition Function

TABLE III
TEST POINT SIZES

Number of Digits in Field Size	Number of Test Points
5	32
6	64
7	128
8	256

Many different fitness functions were attempted, including mean scores produced by the candidate expression tree with a set of test points, and other measures of central tendency. The final fitness function used in this study is inspired by Knuth's work on perfect distribution of hash values produced by hash functions that approximate the golden ratio [25]. The number of test points t used to calculate fitness varies and is assigned based on the number of digits of the field size. These bounds were determined through empirical testing and are summarized in Table III. Equations 4 and 5 show the two calculations comprising the fitness function.

$$TPS = \sum_{n=1}^{t} EvoH(TestPoint[n])(mod\ (L+1)) * \phi \quad (4)$$

$$fitness = \frac{TPS}{t} + penalty \quad (5)$$

Here we seek to minimize the fitness value. A sequence of test-points from the curve under consideration are randomly selected and added to an array called R, which is calculated in line 3 of Algorithm 1. Each test point has the candidate $EvoH$ function applied to it. This creates a sequence of integers in the range 1 to L. This range is maintained by applying a $mod\ L+1$ operation to every point after it has been computed in $EvoH$. This is necessary to ensure that only a valid section number is selected by the hash function. Each value in this sequence is then multiplied by ϕ and summed across all test points resulting in the value TPS found in Equation 4. This value, TPS, is then divided by the number of test points used and a penalty is applied, as shown in Equation 5.

A penalty is applied to the score if a candidate function does not supply enough distinct values when used against the complete collection of test points. This threshold was set to 53% of L and was determined via empirical testing. This penalty discourages candidate functions that return all the same value thereby trivially creating a better fitness.

IV. ANALYSIS

The most direct way to assess the effectiveness of the methodology is to compare the number of iterations required by the evolved hash function compared to the standard hash function in Pollard's Rho Algorithm on the same data. Counting the number of iterations required to solve the ECDLP is the only consistent measure that can be applied between the two different formulations of the Rho algorithm. For example the original version of the algorithm simply solves the ECDLP without any pre-processing of the problem, while

TABLE IV
FIELDS EXAMINED ARRANGED BY NUMBER OF DIGITS

Curve	5 Digits	6 Digits	7 Digits	8 Digits
1	15349	117811	1009807	10184123
2	17027	128813	1142569	16077749
3	18917	167593	1196999	18224243
4	19913	175687	1288657	29151791
5	20731	303679	2297411	34641751
6	29009	389527	3370739	39667153
7	31319	406807	5569079	47102819
8	37117	521393	5689007	47871209
9	42937	783533	8764919	55921661
10	59743	965267	9032839	90050687

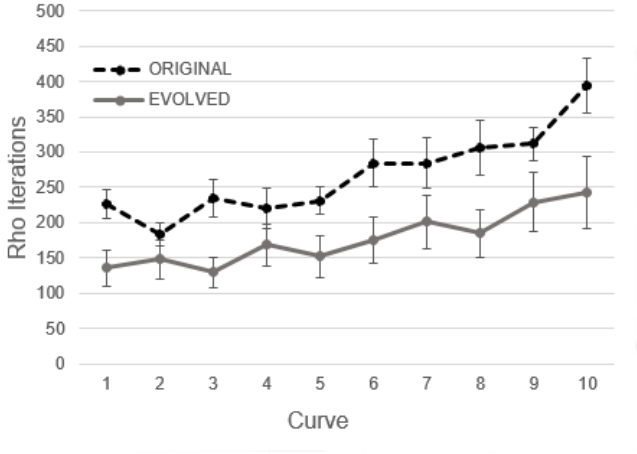

Fig. 4. Number of Iterations Required for the 10 Curves with 5 Digits

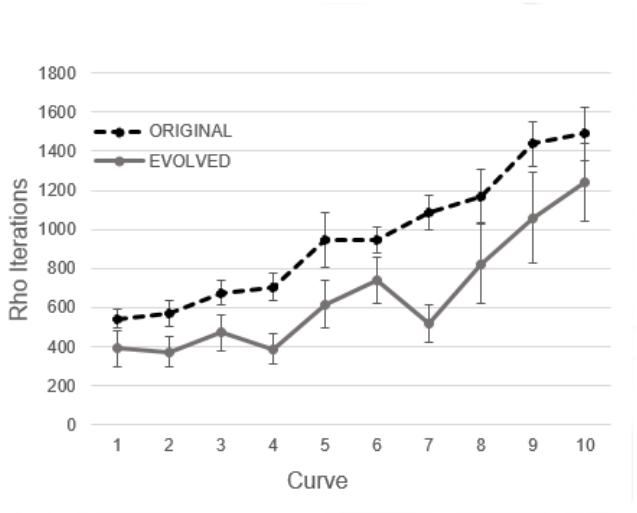

Fig. 5. Number of Iterations Required for the 10 Curves with 6 Digits

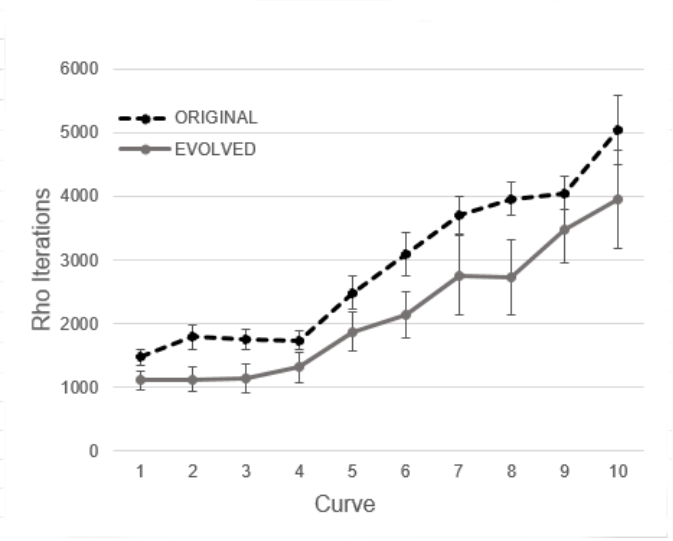

Fig. 6. Number of Iterations Required for the 10 Curves with 7 Digits

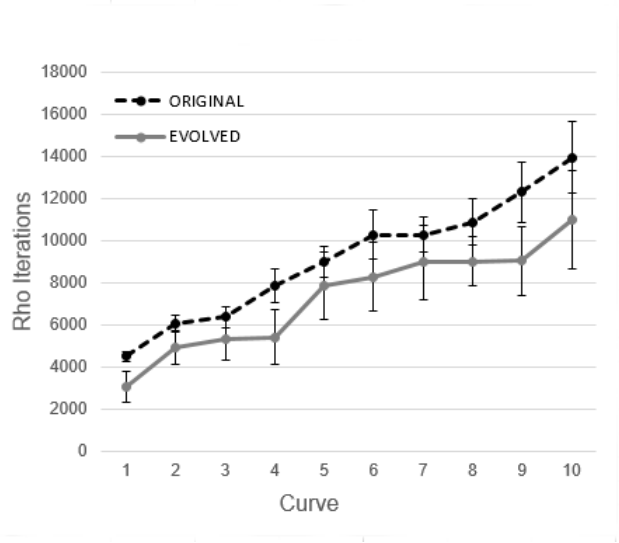

Fig. 7. Number of Iterations Required for the 10 Curves with 8 Digits

in the method proposed by this study a complete genetic programming evolution is computed before the calculation of the ECDLP is even attempted.

The dataset is comprised of 40 different curves of varying sizes, that are summarized in Table IV, where for convenience they are listed by number of digits. Figs. 4, 5, 6, and 7 show the results of these comparisons organized by the number of digits of the field size. For the ten curves presented in each graph the mean score of the original Rho Algorithm based on 30 runs is charted, along with the mean score of 30 runs of the evolved Rho algorithm using the same seed. The curves are arranged by increasing size. A confidence interval of 95% is also presented. The Rho Algorithm using the evolved function clearly results in a lower number of iterations to find the solution versus the original Rho algorithm, for all 40 of the curves. The evolved process consistently turns in significantly fewer overall iterations of the Rho Algorithm to find k although in some cases the confidence interval is larger than for the original process.

During the course of this study many different formulations of the GP were considered that provided less successful results. It was determined that using just the y-coordinate produced better reductions in runtime, as opposed to schemes that used the x-coordinate on its own or in a combination with the y-coordinate. It stands to reason that this might be the case because the y-coordinate moves along the curve quicker and covers a greater range as opposed to the x-coordinate which is symmetric about the x-axis and could possibly confuse the iterating function due to this duplication of values. In a similar vein the introduction of trigonometric functions and random number generators into the GP expression did not decrease runtimes of the Rho algorithm. Similar empirical testing also determined that the inclusion of a protected modulus operator did not aid in decreasing runtime in any way.

A single experiment for a curve of field size 5 digits (or up to approximately 16 bits) involving a complete evolution of 100 generations and a corresponding Rho algorithm generally completed within a few minutes running on a mid-power i5 desktop computer. Curves of field size 6 digits (or approximately 17–20 bits) required a few hours to complete. Curves of field size 7 digits (or approximately 20–24 bits) required days of runtime for a single experiment to complete. Finally, curves of field size 8 digits (or approximately 24–27 bits) completed in about double the amount of time required for those of 7 digits. An important fact worth noting is that the solution to the ECDLP being investigated was found 100% of the time. This perfect success rate was achieved across all 40 curves for each of the 30 runs peformed. Considering all 40 curves and all 30 runs, the evolved Rho algorithm required approximately 71% of the number of iterations compared to the original Rho algorithm. Compare these results to those from [9] which investigated curves of similar size (14, 20 and 32 bits), yet only managed to find the correct solution on average 57% of the time.

These values can also be compared to the Certicom challenge curves [26]. First presented in 2004 the Certicom company introduced a series of ECDLP problems and offered a bounty to researchers who could solve them. This ostensibly was to increase awareness and adoption of elliptic curve cryptosystems. An introductory exercise presented in the white paper proposed that while utilizing a cluster of 3000 computers an expected runtime for a 79 bit curve would be a few hours, an expected runtime of a few days for an 89 bit curve, and an expected runtime of a few weeks for a 97 bit curve. Researchers have found solutions to all three of these exercise curves and to date only one solution for the *Level 1* curves proposed by Certicom has been found. This involved solving a 109 bit curve using a cluster of 2600 computers that ran for 17 months. The method used to find the solution was a modified version of the parallel Rho algorithm.

The curves investigated in this study are not found in

industrial applications. The fields presented in Table IV were chosen for this study to determine if an evolved iterating function decreased runtime of the Rho algorithm; at the current time, choosing larger curves would not allow an adequate number of runs to be completed for each experiment in a reasonable time. With this understanding, however, this initial study provides significant hope that with further study computational intelligence techniques will prove to be a viable option for cryptanalysis of ciphers based on elliptic curves.

V. Conclusion and Future Work

This study presents an initial analysis of how to improve Pollard's Rho Algorithm using computational intelligence techniques. It is shown that replacing the default iterating function with an evolved genetic programming expression reliably causes a reduction in the number of iterations needed to find the solution to the ECDLP.

With this understanding it would be worthwhile to investigate other aspects of the process to determine if it is possible to further decrease the runtime. For example, the number of sections or the value of L could be adjusted for each curve as part of the evolutionary process. The number of test points to use during the evolutionary process could also be examined. This study used a pre-determined set of points for a wide range of cardinalities; there might be some improvement with using a tailored amount of points for each curve examined. It might also be possible to ensure that the test points are distributed evenly across all L sections to help even out the randomness of their occurrence; this study did not account for where on the curve the test points were found. The fitness function might also be investigated to see if a different function or measure of distribution of hash values could yield a better reduction in runtime. Similarly a scheme that tries to randomize point distribution through the R array using other measures of randomness might yield better results. Lastly using a different set of operators in the GP expression tree might produce better results. For example it might be possible to introduce point addition and similar operations that are specific to elliptic curves. Additionally it might be possible to adjust other GP parameters, such as maximum tree size, to determine if there is a positive effect on reducing the number of iterations of the Rho Algorithm.

In conclusion this study shows that there is a strong correspondence between an evolved well distributed iterating function and a reduction in the number of iterations performed by the Rho algorithm.

Acknowledgements

The authors would like to thank Brian Ross and Joseph Brown for their helpful comments and suggestions. The reported study was funded in part by the Natural Sciences and Engineering Research Council of Canada.

References

[1] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, November 1976.

[2] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978. [Online]. Available: http://doi.acm.org/10.1145/359340.359342

[3] V. S. S. Sriram, R. Ramadas, R. Sahay, and G. Sahoo, "Optimizing elliptic curve domain parameters using genetic algorithms," *International Journal of Secure Digital Information Age*, vol. 1, no. 2, 2009.

[4] R. Ratan, "Applications of genetic algorithms in cryptology," in *Proceedings of the Third International Conference on Soft Computing for Problem Solving*. Springer, 2014, pp. 821–831.

[5] B. Delman, "Genetic Algorithms in Cryptology," M.Sc., Rochester Institute of Technology, Rochester, New York, 2004.

[6] E. Y.-T. Ma and C. Obimbo, "An evolutionary computation attack on one-round tea," *Procedia Computer Science*, vol. 6, pp. 171–176, 2011.

[7] B. Ferriman and C. Obimbo, "Solving for the rc4 stream cipher state register using a genetic algorithm," *International Journal of Advanced Computer Science and Applications*, vol. 5, no. 5, pp. 216–223, 2014.

[8] J. Brown, S. Houghten, and B. Ombuki-Berman, "Genetic algorithm cryptanalysis of a substitution permutation network," in *Computational Intelligence in Cyber Security, 2009. CICS '09. IEEE Symposium on*, Mar. 2009, pp. 115–121.

[9] E. C. Laskari, G. C. Meletiou, Y. C. Stamatiou, and M. N. Vrahatis, "Cryptography and Cryptanalysis Through Computational Intelligence," in *Computational Intelligence in Information Assurance and Security*, ser. Studies in Computational Intelligence, N. Nedjah, A. Abraham, and L. d. M. Mourelle, Eds. Springer Berlin Heidelberg, 2007, no. 57, pp. 1–49.

[10] N. Koblitz, "Elliptic Curve Cryptosystems," *Mathematics of Computation*, vol. 48, no. 177, pp. 203–209, Jan. 1987. [Online]. Available: http://www.jstor.org/stable/2007884

[11] V. S. Miller, "Use of Elliptic Curves in Cryptography," in *Lecture Notes in Computer Sciences; 218 on Advances in cryptologyCRYPTO 85*. New York, NY, USA: Springer-Verlag New York, Inc., 1986, pp. 417–426. [Online]. Available: http://dl.acm.org/citation.cfm?id=18262.25413

[12] D. R. Hankerson, S. A. Vanstone, and A. J. Menezes, *Guide to elliptic curve cryptography*. New York : Springer, 2004., 2004.

[13] J. Pollard, "A monte carlo method for factorization," *BIT Numerical Mathematics*, vol. 15, no. 3, pp. 331–334, Sep. 1975.

[14] J. M. Pollard, "Monte Carlo Methods for Index Computation mod p," *Mathematics of Computation*, vol. 32, no. 143, pp. 918–924, Jul. 1978. [Online]. Available: http://www.jstor.org/stable/2006496

[15] A. J. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of applied cryptography*, ser. CRC Press Series on Discrete Mathematics and its Applications. CRC Press, Boca Raton, FL, 1997.

[16] P. van Oorschot and M. J. Wiener, "Parallel Collision Search with Cryptanalytic Applications," *Journal of Cryptology*, vol. 12, no. 1, pp. 1–28, Jan. 1999.

[17] E. Teske, "Speeding up Pollard's rho method for computing discrete logarithms," in *Algorithmic Number Theory*, ser. Lecture Notes in Computer Science, J. P. Buhler, Ed. Springer Berlin Heidelberg, 1998, no. 1423, pp. 541–554. [Online]. Available: http://link.springer.com/chapter/10.1007/BFb0054891

[18] P. Wang and F. Zhang, "Computing elliptic curve discrete logarithms with the negation map," *Information Sciences*, vol. 195, pp. 277–286, Jul. 2012.

[19] F. Zhang and P. Wang, "Speeding up elliptic curve discrete logarithm computations with point halving," *Designs, Codes and Cryptography*, vol. 67, no. 2, pp. 197–208, May 2013.

[20] P. Wang and F. Zhang, "Improving the Parallelized Pollard Rho Method for Computing Elliptic Curve Discrete Logarithms," in *2013 Fourth International Conference on Emerging Intelligent Data and Web Technologies (EIDWT)*, Sep. 2013, pp. 285–291.

[21] ——, "An Efficient Collision Detection Method for Computing Discrete Logarithms with Pollard's Rho," *Journal of Applied Mathematics*, pp. 1–15, Jan. 2012.

[22] ——, "Improved Pollard rho method for computing discrete logarithms over finite extension fields," *Journal of Computational and Applied Mathematics*, vol. 236, pp. 4336–4343, Nov. 2012.

[23] J. R. Koza, *Genetic programming : on the programming of computers by means of natural selection*, ser. Complex adaptive systems. Cambridge, Mass. MIT Press, 1992, a Bradford book. [Online]. Available: http://opac.inria.fr/record=b1082356

[24] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné, "DEAP: Evolutionary algorithms made easy," *Journal of Machine Learning Research*, vol. 13, pp. 2171–2175, jul 2012.

[25] D. E. Knuth, *The Art of Computer Programming, Volume 3: (2Nd Ed.) Sorting and Searching*. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 1998.

[26] (2009) Certicom ecc challenge. [Online]. Available: http://www.certicom.com/images/pdfs/challenge-2009.pdf