Pollard's Rho Attack

Paweł Murdzek, Jakub Włodarz, Patryk Średniawa, Piotr Szewczyk,
Bui Giang Nam, Piotr Gutowski, Zuzanna Popławska
Institute of Telecommunications, Warsaw University of Technology

***Abstract*—This paper presents an analysis of Pollard's Rho attack, a generic algorithm used to solve the Discrete Logarithm Problem (DLP). The work focuses on the application of this method in the context of Elliptic Curve Cryptography (ECC), specifically for curves defined over binary fields (F_{2^m}).**

The first part of the paper discusses the necessary mathematical foundations in detail, including the principles of modular arithmetic in binary fields and the key properties of elliptic curves. Subsequently, the paper analyzes the source of vulnerability, explaining how the cyclicity of a subgroup of points on an elliptic curve is exploited by attacking algorithms.

The main part of the article is dedicated to the Elliptic Curve Discrete Logarithm Problem (ECDLP), which constitutes the foundation of ECC system security. A detailed specification of Pollard's Rho attack is presented, discussing its principles of operation, computational complexity, and practical limitations.

The final section presents a practical implementation of the attack. It includes both the general scheme of the algorithm in the form of pseudocode and its specific realization in C++. The work concludes with a presentation and discussion of the results obtained during the testing of the implemented algorithm.

I. Mathematical Theory

A. Description of Modular Arithmetic for Binary Fields

Modular arithmetic plays a key role in computer science and cryptography using finite fields. This article focuses on a specific type: binary fields (denoted as $\mathbf{GF}(2^n)$ or $\mathbf{F}_{2^n}$). These are interesting due to the ease of implementation in hardware solutions.

Addition corresponds to the XOR operation:

$$\begin{aligned} 0+0 &= 0 \pmod 2 \qquad & 0+1 &= 1 \pmod 2 \\ 1+0 &= 1 \pmod 2 \qquad & 1+1 &= 0 \pmod 2 \end{aligned}$$

It is worth noting that addition is its own inverse ($a + a = 0$).

Multiplication corresponds to the AND operation:

$$\begin{aligned} 0\cdot 0 &= 0 \pmod 2 \qquad & 0\cdot 1 &= 0 \pmod 2 \\ 1\cdot 0 &= 0 \pmod 2 \qquad & 1\cdot 1 &= 1 \pmod 2 \end{aligned}$$

The inverse element of 1 is 1.

1) Extended Binary Fields $\mathbf{GF}(2^n)$*:* In cryptography, for computational purposes, larger sets of elements are often used. These elements are represented as polynomials with coefficients from $\mathbf{GF}(2)$ and a degree less than n. The set of these polynomials constitutes the 2^n elements of the field $\mathbf{GF}(2^n)$. Arithmetic operations are performed on polynomials, and the result is then reduced modulo a certain irreducible polynomial of degree n. Addition is performed using bitwise XOR on the binary representations of the elements.

Multiplication, on the other hand, is performed in two steps. It begins with standard multiplication $A(x) \cdot B(x)$. Then, the result constitutes the remainder of the division by a fixed irreducible polynomial $P(x)$ of degree n with coefficients from $\mathbf{GF}(2^n)$. This remainder is the result of modular multiplication:

$$A(x) \cdot B(x) \equiv R(x) \pmod{P(x)}$$

It is worth mentioning that the polynomial $P(x)$ acts as the modulus and ensures that the result remains in $\mathbf{GF}(2^n)$. For every non-zero element $A(x) \in \mathbf{GF}(2^n)$, there exists an inverse element $A^{-1}(x)$ such that:

$$A(x) \cdot A^{-1}(x) \equiv 1 \pmod{P(x)}$$

To find this inverse, the Extended Euclidean Algorithm for polynomials is used.

The computational efficiency of modular arithmetic largely stems from the fact that operations can be implemented directly on processor bits. For this reason, fields $\mathbf{GF}(2^n)$ are widely used in hardware implementations and cryptography.

Algorithms implementing the above operations will be described in Section V.

B. Properties of Elliptic Curves

Algebraically, an elliptic curve E defined over a field K is a smooth (non-singular) cubic algebraic curve in the projective plane $P^2(K)$, which possesses at least one K-rational point. This point is typically designated as the identity element of the group.

For practical purposes, provided the characteristic of the field K is different from 2 and 3 (which is the case e.g., for $K = \mathbb{R}$, $K = \mathbb{Q}$ or finite fields $\mathbb{F}_p$ for $p > 3$), every elliptic curve can be reduced, through appropriate coordinate transformation, to the generalized Weierstrass form:

$$E : y^2 = x^3 + Ax + B$$

for coefficients $A, B \in K$.

A key condition is the non-singularity of the curve, which guarantees that the curve has no "cusps" or self-intersections. This condition is equivalent to the non-vanishing of the curve's discriminant:

$$\Delta = -16(4A^3 + 27B^2) \neq 0$$

When $\Delta = 0$, the curve is singular and does not possess the group structure of interest [**?**].

The set of K-rational points on the curve E is defined as:

$$E(K) = \{(x, y) \in K \times K \mid y^2 = x^3 + Ax + B\} \cup \{O\}$$

The element O is a special, distinguished point called the point at infinity. Its inclusion is necessary for the algebraic closure of group operations [**?**].

The most important property of the set of points $E(K)$ is that it forms an abelian group with respect to the point "addition" operation. This operation is defined geometrically:

1) **Identity Element:** This is the point at infinity O. For any point P on the curve:
$$P + O = P$$
2) **Inverse Element:** For a point $P = (x, y)$, its inverse element is the point $-P = (x, -y)$, which is its geometric reflection across the OX axis.
3) **Addition $P + Q$ (when $P \neq Q$):** To add two distinct points P and Q, we draw a line through them. According to Bézout's theorem, this line will intersect the cubic curve at exactly three points [**?**]. Let the third point of intersection be R. The sum $P+Q$ is defined as $-R$ (the reflection of R across the OX axis).
$$P + Q = -R$$
4) **Doubling $P+P$:** Instead of a secant through two points, we draw a tangent line to the curve at point P. This tangent will intersect the curve at one more point R. The sum is defined as $-R$.
$$P + P = -R$$
5) **Sum $P+(-P)$:** A line passing through $P = (x, y)$ and $-P = (x, -y)$ is a vertical line. Such a line does not intersect the curve at a third point in the affine plane – it "intersects" it at the point at infinity O. Thus, $R = O$. By definition, $P + (-P) = -O$. Since O lies "on the axis of symmetry" (one can think of it as a point on a vertical line at infinity), its reflection is itself, i.e., $-O = O$.
$$P + (-P) = O$$

This geometric law of composition satisfies all group axioms (associativity, commutativity, identity element, inverse element), making $E(K)$ one of the fundamental structures in number theory.

II. Source of the Attack - Subgroup Cyclicity

The security of Elliptic Curve Cryptography (ECC) relies on the hardness of the Discrete Logarithm Problem (DLP) within a finite cyclic subgroup. Multiplying a point P on a curve in a finite field inevitably leads to repeating results, or "looping" (e.g., $5P = 0$, followed by $6P = 1P$). These repeating points form a cyclic subgroup, which is described as the "foundation of elliptic curve cryptography and an important aspect of cryptanalysis" [**?**].

Pollard's Rho attack is a fundamental example of how this property is exploited in cryptanalysis. It is a *generic attack*, meaning it works on any finite cyclic group regardless of its construction (whether they are points on a curve or numbers in a multiplicative group) [**?**].

The relationship between cyclicity and the attack can be explained as follows [**?**]:

1) **Finite Space:** A cyclic subgroup, by definition, has a finite order (number of elements), which we will denote as n. This means there are only n unique points that can be generated by multiplying the generator P.
2) **Pseudorandom Walk:** Pollard's Rho attack involves performing a "pseudorandom walk" through the points of this subgroup. Starting from a certain point, a sequence of subsequent, seemingly random points is generated using an iterating function.
3) **Inevitable Collision (Birthday Paradox):** Since the number of points in the subgroup is finite (n), any sequence of generated points must eventually begin to repeat. This is a direct result of the Dirichlet box principle (pigeonhole principle). Furthermore, according to the so-called **birthday paradox**, one statistically expects to find a collision (a point that can be reached in two different ways) much faster than the order of the entire group. Instead of n steps, a collision is expected after approximately $\mathcal{O}(\sqrt{n})$ steps [**?**]. The birthday paradox itself illustrates that in a set of n elements (e.g., days in a year), the probability of finding a collision (two people with the same birthday) exceeds 50% with a selection of only about $\sqrt{n}$ elements (in the case of 365 days, just 23 people are sufficient). In Pollard's Rho attack, the "days of the year" are the points in the subgroup, and the "people" are the steps of the algorithm [**?**].
4) **Shape of "Rho" (ρ):** The name of the algorithm comes from the fact that the sequence of generated points resembles the Greek letter Rho (ρ) — it consists of a "tail" (points visited before the first collision) and a "loop" (the cycle into which the sequence falls).

In summary, "subgroup cyclicity" is the fundamental property that guarantees Pollard's Rho attack will work at all. It is the finiteness and cyclic nature of the group that ensure a "random walk" must eventually find a cycle, and the birthday paradox dictates that finding this collision is computationally feasible (with square root complexity) for groups of insufficiently large order.

III. Discrete Logarithm Problem

The classical Discrete Logarithm Problem is based on determining the value *x*, also referred to as the discrete logarithm of *h* to the base *g*, satisfying the equality:

$$a^x \equiv h \pmod{p}$$

where a is the base, x is the discrete logarithm, h is the remainder, and p is the modulus.

In cryptographic applications, this problem is utilized due to the occurring computational asymmetry. Modular exponentiation $g^x \ (mod\ p)$ is significantly easier than searching for the value *x*. Thanks to this property, it is possible to effectively secure, among other things, public keys.

For the attack in question, it is necessary to transfer this problem to elliptic curves, which changes the entire process. The main element of the algorithm is the multiplication of a given point by a scalar defined by the appropriate equation:

$$Q = l * P$$

where Q is the public key, l is the private key, and P is the base point.

In this case, obtaining the public key is relatively simple and involves multiplying the private key and the point. However, attempting to determine the private key in the same way is exceptionally computationally expensive.

The above solution aims to avoid using large numbers by performing all operations on points of the elliptic curve. Such a change allows offering a similar level of security using a shorter key.

Until recently, methods from the family of elliptic curve cryptography algorithms were considered impossible to break by current computers. Pollard's Rho attack is described as the first real threat to this field of cryptography [**?**].

IV. Attack Specification and Limitations

The security of cryptography based on elliptic curves stems from the difficulty of solving the discrete logarithm problem in a cyclic subgroup of the curve. The attacker aims to find the private key l, knowing the point P and the corresponding public key $Q = lP$. Security results from the limitation in the efficiency of determining l.

1) Attack Specification: Pollard's Rho attack exploits the properties of finite cyclic groups in a way that allows for a radical reduction in the time required to find a solution [**?**]. A key element of the attack is an iterative function (or pseudorandom function), enabling the traversal of curve points over elements of the group $E(K)$ in a deterministic manner.

For each point X in the sequence, the next point X' is calculated as:

$$X' = f(X)$$

The function $f(X)$ is designed so that each point X in the sequence is represented as a linear combination of the generator P and the public key Q, where the coefficients a and b are updated in each step:

$$X = a \cdot P + b \cdot Q$$

The attack does not search for the key l directly. At some point, a **collision** occurs in the sequence, meaning two different iterative sequences i and j lead to the same point $X_i = X_j$.

Since $Q = l \cdot P$, a collision implies that:

$$a_i \cdot P + b_i \cdot Q = a_j \cdot P + b_j \cdot Q$$

Substituting $Q = l \cdot P$ and grouping with respect to P, we obtain:

$$(a_i + b_i \cdot l) \cdot P = (a_j + b_j \cdot l) \cdot P$$

Since the order of the group is n, the equality of points implies the equality of scalars modulo n:

$$a_i + b_i \cdot l \equiv a_j + b_j \cdot l \pmod{n}$$

This allows calculating the value of l by analyzing the difference between the sequences. Transforming the above equation, one can **determine the private key** l:

$$l \cdot (b_i - b_j) \equiv a_j - a_i \pmod{n}$$

The value l is calculated using the *Extended Euclidean Algorithm* to find the inverse of $(b_i - b_j)$ modulo n.

It is important to correctly detect the collision. For this purpose, **Floyd's Cycle-Finding** algorithm, also known as the "tortoise and hare" method, is used. This algorithm involves simultaneously tracking two steps of points — one moving by one step (X_{i+1}), while the other moves by two steps (X_{i+2}). When both points meet ($X_i = X_{2i}$), it means a collision has been found.

There are various variants of the attack:

- **Pollard's Rho with distinguished points** — a method improving collision detection,
- **Pohlig–Hellman** — utilizes the factorization of the subgroup order,
- **Baby-step Giant-step** — characterized by similar complexity but higher memory consumption.

2) Attack Limitations: Limitations do not result from a lack of efficiency, but rather from the necessity of minimizing risk through the proper selection of ECC parameters. The greatest limitations include:

- Time complexity $O(\sqrt{n})$, where n is the order of the cyclic subgroup [**?**]. If the subgroup order is a large prime number, none of the methods offer a practical attack. This is the upper bound on the efficiency of generic DLP attacks.
- Curves with improper parameters (e.g., anomalous or with a small cofactor) may be susceptible to specialized attacks reducing ECDLP to simpler problems.
- Most effective attacks require simultaneous mathematical weaknesses and implementation errors.

V. Pollard's Rho Attack Pseudocode

Pollard's Rho algorithm was implemented using the C++ programming language. Realizing the attack as a program running on a CPU was not an optimal solution in terms of performance, but it provided a good introduction to issues related to cryptanalysis algorithms. Additionally, C++ software can be accelerated with hardware accelerators, e.g., using the OpenCL platform.

A. Algorithms Used in Calculation Implementation

The project implementation assumes a 32-bit architecture, meaning each vector representing polynomials is organized into 32-bit blocks. The field F_{2^m} having elements of order at most $m - 1$ will be represented as an array of 32-bit binary vectors. The size of the array, denoted as t, is $\lceil m/32 \rceil$. The algorithms used were taken from the textbook *Guide to Elliptic Curve Cryptography* [**?**].

1) Addition: Addition is trivial; it requires adding all vectors to each other using the XOR operation. Algorithm 1 presents the steps needed for polynomial addition.

Algorithm 1 Polynomial Addition Algorithm

Input: polynomials a(z) and b(z) of degree m-1
Output: c(z) = a(z) + b(z)

1: **for** i from 0 to t-1 **do**
2: $C[i] \leftarrow A[i] \oplus B[i]$
3: **end for**
4: Return(C)

Algorithm 2 Polynomial Multiplication Algorithm

Input: polynomials a(z) and b(z) of degree m-1
Output: c(z) = a(z) · b(z)

1: $C \leftarrow 0$
2: **for** k from 0 to 31 **do**
3: **for** j from 0 to t-1 **do**
4: **if** a[j][k] == 1 **then**
5: C[j] ← C[j] ⊕ B[j]
6: **end if**
7: **end for**
8: **if** k $\neq 31$ **then**
9: B ← B · z
10: **end if**
11: **end for**

2) Multiplication: In Algorithm 2, multiplication performed on polynomial B by polynomial z means a bitwise shift of one bit to the right.

Algorithm 3 Polynomial Squaring Algorithm for 32-bit Architecture

Input: polynomial a(z) of degree m-1
Output: c(z) = a(z)^2

1: For each byte $d = (d_7, ..., d_2, d_1)$ calculate the 16-bit value $T(d) = (0, d_7, ..., 0, d_1, 0, d_0)$
2: **for** i from 0 to t-1 **do**
3: A[i] = (u_3, u_2, u_1, u_0) where u_j is a byte
4: C[2i] ← (T(u_1), T(u_0))
5: C[2i+1] ← (T(u_3), T(u_4))
6: **end for**

3) Squaring: The polynomial squaring algorithm (presented in Algorithm 3) involves inserting zeros between the bits of each vector in the array representing the polynomial.

4) Reduction: The algorithm for reducing a polynomial obtained by multiplication or exponentiation is found in the description of Algorithm 4.

Algorithm 4 Polynomial Reduction Algorithm

Input: polynomial c(z) of degree 2m-2, reduction polynomial f(z) = z^m + r(z)
Output: c(z) mod f(z)

1: **for** k from 0 to 31 **do**
2: Calculate $u_k(z) = z^k r(z)$
3: **end for**
4: **for** i from 2m-2 to m **do**
5: **if** c_i = 1 **then**
6: j = $\lfloor (i-m)/32 \rfloor$, k=(i-m) - 32*j
7: C[j] ← C[j] ⊕ u_k(z)
8: **end if**
9: **end for**
10: Return C

Algorithm 5 Polynomial Inversion Algorithm

Input: polynomial a(z)
Output: a^{-1} mod f

1: u ← a, v ← f
2: $g_1 \leftarrow 1$, $g_2 \leftarrow 0$
3: **while** u $\neq 1$ **do**
4: j ← degree(u) - degree(v)
5: **if** j < 0 **then**
6: u ↔ v, $g_1 \leftrightarrow g_2$, j ← -j
7: u ← u + z^{j}gv
8: $g_1 \leftarrow g_1$ + z$^j g_2$
9: **end if**
10: **end while**
11: Return(g_1)

5) Inversion: Inversion using the Extended Euclidean Algorithm is presented in Algorithm 5.

VI. Pollard's Rho Algorithm

Algorithm 6 describes the pseudocode of Pollard's Rho factorization algorithm for elliptic curves.

Algorithm 6 Pollard's Rho Algorithm

generator G and point P
natural number x satisfying P = xG

Initialization:
$i \leftarrow 0$
$a_0, b_0 \leftarrow$ natural random numbers between 2 and group order n
$X_0 \leftarrow O$
while i $\leq$ n, where n is the group order **do**
    Calculate a_i, b_i
    Calculate a_{2i}, b_{2i}
    Calculate $X_i = a_i G + b_i P$
    Calculate $X_{2i} = a_{2i} G + b_{2i} P$
    if $X_i = X_{2i}$ **then**
        if $b_i = b_{2i}$ **then**
            Restart algorithm
        else
            Return $x = (a_i - a_{2i})(b_{2i} - b_i)^{-1} \mod n$
        end if
    else
        i = i+1
    end if
end while

A. Functions Generating Parameters in the Next Iteration of Pollard's Rho Algorithm

The functions in Algorithm 6 generating values a_i, b_i, a_{2i}, b_{2i} are executed based on the x-coordinate of the current calculated point X_i. The entire x-axis space is divided into 3 groups by performing modulo 3 on the x-coordinate.

Depending on the result of the modulo 3 operation, X_i equals:

$$X_{i+1} \begin{cases} X_i + Q & \text{for } x \mod 3 = 0 \\ 2 \cdot X_i & \text{for } x \mod 3 = 1 \\ X_i + P & \text{for } x \mod 3 = 2 \end{cases} \tag{1}$$

Whereas coefficients a_{i+1} and b_{i+1} are generated as follows:

$$a_{i+1} \begin{cases} a & \text{for } x \mod 3 = 0 \\ 2 \cdot a \mod f(x) & \text{for } x \mod 3 = 1 \\ a + 1 \mod f(x) & \text{for } x \mod 3 = 2 \end{cases} \tag{2}$$

$$b_{i+1} \begin{cases} b + 1 \mod f(x) & \text{for } x \mod 3 = 0 \\ 2 \cdot b \mod f(x) & \text{for } x \mod 3 = 1 \\ b & \text{for } x \mod 3 = 2 \end{cases} \tag{3}$$

Where n is the curve order.

VII. Results

The C++ implementation was not realized due to difficulties in creating a data structure in the form of bit arrays. The maximum value for which a working implementation was successfully programmed was 32 bits.

For comparison, an equivalent in the Python programming language was created, where there is no need to worry about issues with implementing appropriate data structures. However, such a degree of freedom is obtained at the cost of performance and limited control over computational resources used during calculations.

A. Selected Curve

Using the ecgen tool [**?**], a given elliptic curve domain was generated:

- degree m = 16
- coefficient a = 0x2905
- coefficient b = 0x886f
- irreducible polynomial: $f(x) = x^{16} + x^5 + x^3 + x + 1$
- curve order n = 65920
- generator: G = (0xba04, 0x9b3b)

It is worth noting the curve order, as it is not a prime number. It was not possible to generate a curve with a prime order despite several hours of ecgen program operation. The curve order will be significant for the results.

B. Attack Execution

To speed up the collision detection algorithm, the number of iterations was limited to the square root of the curve order (theoretical computational complexity of Pollard's Rho algorithm). Typically, the program needed 3-4 executions of Pollard's Rho to find a collision. It is worth mentioning that Pollard's Rho algorithm does not always have to return a result due to its random nature.

In many cases, despite finding a collision, the result was incorrect. The result formula described in Algorithm 6 requires calculating the inversion of $(b_{2i} - b_{2i}) \mod n$, so $gcd((b_{2i} - b_{2i}), n) == 1$. For n that are not prime numbers, it is difficult to obtain a result satisfying this condition. For curves with orders that are not prime numbers, Pollard's Rho often terminates unsuccessfully.

VIII. Summary

This paper discusses Pollard's Rho Attack as an algorithm for solving the discrete logarithm problem for elliptic curves over binary fields F_{2^m}. Chapter I discussed the mathematical foundations: arithmetic in binary fields, properties of elliptic curves, and the resulting group structure. From the properties of elliptic curves, it follows that the source of the attack's effectiveness is the cyclicity of the subgroup and inevitable collisions resulting from the birthday paradox.

Arithmetic on binary fields and Pollard's Rho algorithm were implemented in the Python programming language. It was not possible to conduct an attack on the curve generated in the ecgen program due to the curve order not being a prime number.