# Cryptography Based on 2D Ray Tracing

Sneha Mohanty and Christian Schindelhauer

November 20, 2024

# CRYPTOGRAPHY BASED ON 2D RAY TRACING

TECHNICAL REPORT

**Sneha Mohanty**
Department of Computer Networks and Telematics
University of Freiburg
Freiburg, Germany
mohanty@informatik.uni-freiburg.de

**Christian Schindelhauer**
Department of Computer Networks and Telematics
University of Freiburg
Freiburg, Germany
schindel@informatik.uni-freiburg.de

November 20, 2024

## ABSTRACT

We introduce a novel symmetric key cryptographic scheme involving a light ray's interaction with a 2D cartesian coordinate setup, several smaller boxes within this setup, of either reflection or refraction type and $1^{st}$, $2^{nd}$ or $3^{rd}$ degree polynomial curves inside each of these smaller boxes. We also incorporate boolean logic gates of types XOR, NOT-Shift and Permutation which get applied to the light ray after each interaction with a reflecting or refracting polynomial curve. This alternating interaction between Optical gates (polynomial curves) and Non-optical gates creates a complex and secure cryptographic system. Furthermore, we design and launch customized attacks on our cryptographic system and discuss the robustness of it against these.

***Keywords*** Cryptography · Polynomial objects · Plaintext · Ciphertext · Key

## 1 Introduction

We introduce the first ever symmetric key cryptographic system involving a two-fold interaction of a light ray with objects in a 2D (x,y)-cartesian coordinate setup and its projection using boolean gates (XOR, NOT-Shift and Permutation). We also formulate and launch customized attacks on our Cryptographic system. We draw inspiration for our work mainly from the paper by Reif et al. [1994], wherein a light ray begins at a certain position and depending on the configurations of various objects in the 3D setup (optical system), it is determined whether or not the final light ray exits at a fixed point, $p$.

We found it interesting that a light ray could be used to encrypt and decrypt sensitive information in a given 2D setup instead of using textual, sound or even image based messages.

## 2 Related Work

As mentioned in the previous section, our work is inspired mainly from Reif et al. [1994].It has been concluded that out of the six different combinations of optical systems that have been illustrated in this paper, except for two of the simplest configurations, the ray tracing problem in 3D is undecidable. Han et al. [1999] worked on Optical image Encryption based on XOR operations. Blansett et al. [2003] from the Sandia National Laboratories in the US discuss the Photonic Encryption using All Optical Logic. In this paper, cryptographic algorithms have been examined in detail and the constraints of optical logic gate technology have been determined. In addition, novel encryption approaches that utilize photonic properties (such as; dispersion, polarization, etc.) that could be modulated by certain electrical devices have been explored. The illumination problem is discussed in Tokarsky [1995] regarding Polygonal rooms where they use right, acute and obtuse isosceles triangles mapped throughout the room to show that not every point is illuminable from every other point within this closed space.

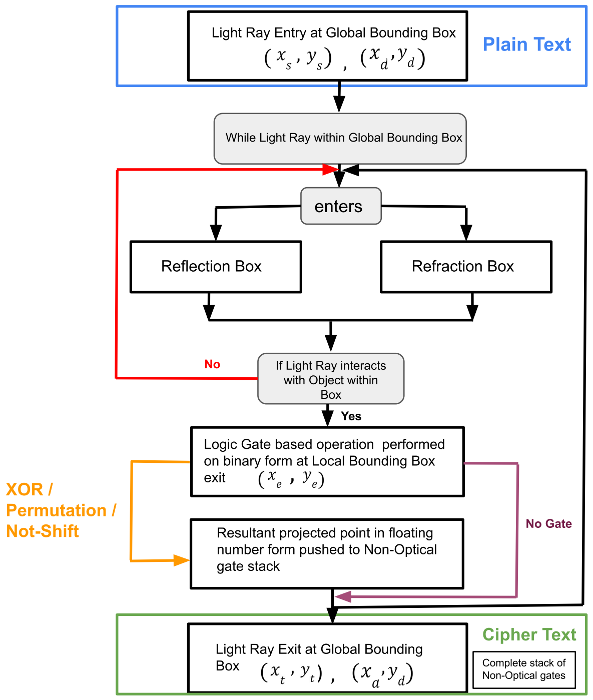

Light Ray Entry at Global Bounding Box
$(x_s, y_s)$ , $(x_d, y_d)$
Plain Text
While Light Ray within Global Bounding Box
enters
Reflection Box
Refraction Box
No
If Light Ray interacts with Object within Box
Yes
Logic Gate based operation performed on binary form at Local Bounding Box exit $(x_e, y_e)$
XOR / Permutation / Not-Shift
No Gate
Resultant projected point in floating number form pushed to Non-Optical gate stack
Light Ray Exit at Global Bounding Box $(x_t, y_t)$ , $(x_d, y_d)$
Cipher Text
Complete stack of Non-Optical gates

Figure 1: Overview of the Cryptographic system

## 3 Overview

The symmetric key cryptographic system consists of a 2D cartesian coordinate setup, with a Global Bounding Box and several smaller Local Bounding Boxes. The Local Bounding Boxes are of two types, i.e; Reflection (black in color) as well as Refraction (red in color). Each of the Local Bounding Boxes has atmost one polynomial curve inside it, of either $1^{st}$, $2^{nd}$ or $3^{rd}$ degree. These Local Bounding boxes are rotated and translated with respect to the origin $(0, 0)$ and are therefore scattered across the Global Bounding Box. Besides these, Non-optical boolean gates such as; XOR, NOT-shift as well as permutation are also part of the scheme. The Plaintext of our scheme consists of the initial $(x_s, y_s)$-position of the light ray at the Global Bounding Box as well as the direction $(dx_s, dy_s)$ at this entry point. The Ciphertext consists of the final $(x_t, y_t)$-position of the light ray, the direction $(dx_t, dy_t)$ at the exit point of the Global Bounding Box as well as the stack of Non-optical gate boxes. The Key of the scheme consists of the Global Bounding Box parameters, the crypto-element as well as object box parameters of the individual curves inside the reflection as well as refraction Local Bounding Boxes. An Overview of the Cryptographic system has been shown in Figure 1. More details about the Components of the Optical system, the Non-optical gate system as well as the Cryptographic scheme are elaborated in Section 4, Section 5 as well as Section 6 respectively.

A Key as well as the sequence of a sample encryption have been illustrated in Figure 2. We use high precision keys (512 bits), in order to render a seamless encryption-decryption process without any data losses.

Furthermore, we also formulate attacks on our cryptographic system. These have been covered in more detail in the Section 7. We have shown that our cryptographic system is very robust against attacks in general and requires special strategies even to partially tackle it.

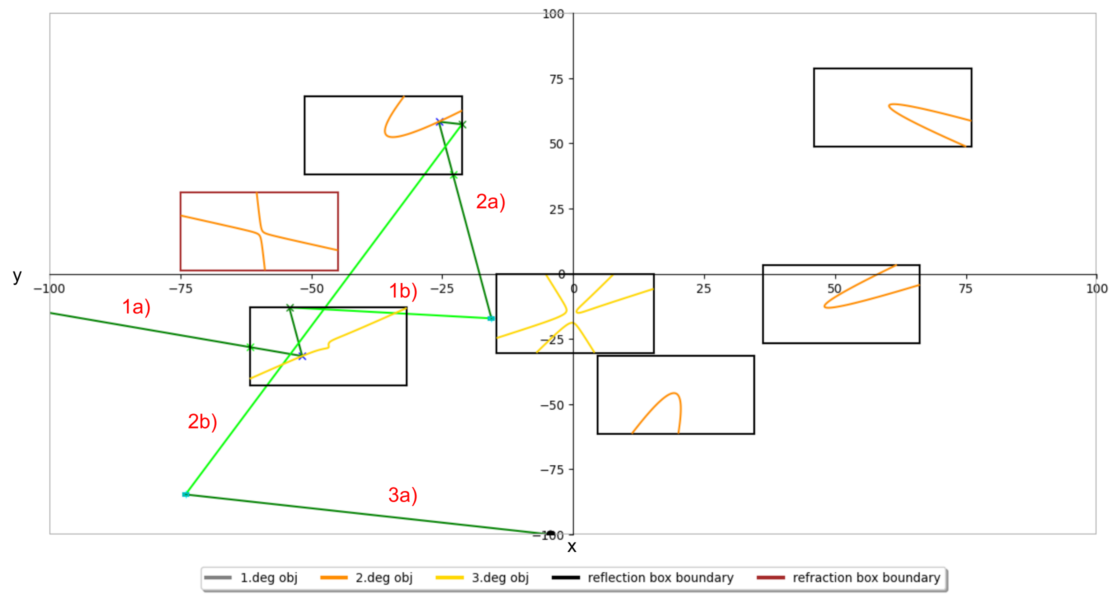


Figure 2: Sample Key and Encryption on the Key

## 4 Components of the Optical System

### 4.1 Global Bounding Box

The Global Bounding Box is a 2D box in cartesian (x,y) coordinates. It contains all the Local Bounding Boxes as well as the Non-Optical gates.

### 4.2 Local Bounding Box

The Local Bounding Boxes are the smaller boxes contained within the Global Bounding Box, of either Reflection (black colored) or Refraction type (red colored). Each Local Bounding Box contains at most one curve, either of $1^{st}$, $2^{nd}$ or $3^{rd}$ degree type.

### 4.3 Optical gates

The optical system consists of a light ray, $1^{st}$, $2^{nd}$ and $3^{rd}$ degree polynomials enclosed by their individual Local Bounding Boxes. Objects contained within the Local Bounding Boxes could be of three types, i.e; $1^{st}$, $2^{nd}$ or $3^{rd}$ degree. While the $1^{st}$ degree objects are parts of a polygon, $2^{nd}$ degree curves could be conic sections such as; parabolae, hyperbolae, ellipses, spheres or generic $2^{nd}$ degree curves. These are summarized in Table 1. The equations of the different curves used in our cryptographic scheme as well as the transformed (rotated as well as translated) coordinates, $(X, Y)$ of the curves within the Global Bounding Box, are shown in Eq. (1).

Table 1: **List of Conic Sections in 2D**

| Polynomial Curve Type | Equation |
|---|---|
| Parabola | $ax^2 + bx + c = y$<br>$a(x-h)^2 + k - y = 0$ |
| Ellipse | $x^2/a^2 + y^2/b^2 = 1$<br>$b^2x^2 + a^2y^2 - a^2b^2 = 0$ |
| Hyperbola | $x^2/a^2 - y^2/b^2 = 1$<br>$b^2x^2 - a^2y^2 - a^2b^2 = 0$ |

$$Dx + Ey + F = 0$$

$$Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0$$
$$Gx^3 + Hy^3 + Kx^2y + Lxy^2 + Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0$$

$$R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

$$T = \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} x - x_0 \\ y - y_0 \end{bmatrix}$$

$$\begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x - x_0 \\ y - y_0 \end{bmatrix} \tag{1}$$

### 4.3.1 Light Ray

This light ray is a vector, with the form described by Eq. (2), that interacts with the objects in the 2D environment and thereafter in each case, behaves as a reflected or refracted ray depending on which type of Local Object Box (described in Section 4.2) it has entered. A light ray is an element that has a source point $(x_s, y_s)$ and a direction $(x_d, y_d)$. We use the vector representation of a line to describe a light ray as follows :

$$\begin{pmatrix} x_s \\ y_s \end{pmatrix} + \lambda \begin{pmatrix} x_d \\ y_d \end{pmatrix} \tag{2}$$

The light ray could be described by a linear equation. We chose the vector representation due to the limitation that the light ray only travels in one direction. The use of the vector representation leads to a negative $\lambda$ if the intersection of the light ray and an object is *behind* the source of the light ray. The positive $\lambda$ and negative $\lambda$ are derived from solving linear, quadratic and cubic equations in our case, as illustrated in the following sections.

### 4.3.2 Tangent and Normal

**Tangent** The slope of the tangent is calculated by taking the $\frac{dy}{dx}$. We take a point,($x_t$,$y_t$) of interest on the tangent. The intercept, $c_t$ of the tangent line can then be calculated by solving the equation of the tangent with the aforementioned slope and point coordinates in consideration.

**Normal** The slope of the normal is the calculated as $-\frac{dx}{dy}$. Similar to the above, the intercept, $c_n$ of the normal line can be calculated by solving the equation of the normal by taking the aforementioned slope of the normal and point of interest, i.e; ($x_n$,$y_n$). The slope of the normal is valid only as long as the slope of the tangent, $\frac{dy}{dx} \neq 0$. The normal, $\vec{n}$, can also be calculated by taking the gradient of the object, $\nabla F$, at the point of intersection. The resulting vector is not limited by the individual values of $dx$ nor $dy$ and may be oriented in any direction. For consistency, the normal vector is set to point 'into' the object. More specifically, $\vec{i} \cdot \vec{n} \geq 0$. When this is not the case, $\vec{n}$ is set to $-\vec{n}$.

### 4.3.3 Intersection Point

This section illustrates the intersection of the light ray with various objects. The idea would be to incorporate Eq.(2) into the equations of the objects in 2D, shown in Eq. (1) based on which object the light ray is intersecting. This would then result in equations with coefficients of $\lambda$, $\lambda^2$ and/or $\lambda^3$ depending on the degree of the object (polynomial).

**1st degree** The first order equations involve linear components and therefore an intersection of the light ray with them would entail solving the linear equation in $\lambda$ as shown in Eq. (3).

$$a\lambda + b = 0 \tag{3}$$

**2nd degree** We can compute the point of intersection between the light ray and the second degree object by substituting the x and y values of the object before they are rotated, with the light ray values as illustrated in Eq. (4).

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} cos\theta & sin\theta \\ -sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} (x_s + \lambda x_d) - h \\ (y_s + \lambda y_d) - k \end{bmatrix} = \begin{bmatrix} cos\theta & sin\theta \\ -sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} x_s - h \\ y_s - k \end{bmatrix} + \lambda \begin{bmatrix} cos\theta & sin\theta \\ -sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} x_d \\ y_d \end{bmatrix} \tag{4}$$

$$a\lambda^2 + b\lambda + c = 0$$

$$\lambda_{1/2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

| discriminant | number of real roots |
|---|---|
| $\Delta > 0$ | 2 |
| $\Delta = 0$ | 1 |
| $\Delta < 0$ | 0 |

$$\Delta = b^2 - 4ac$$

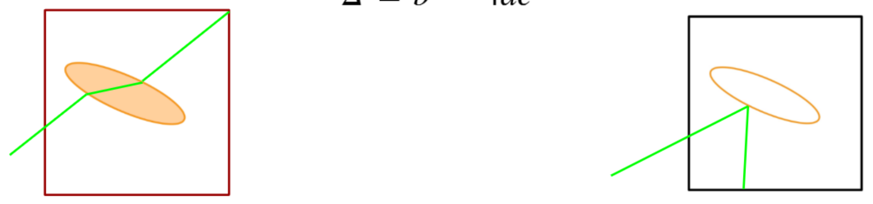


Figure 3: The Light ray vector intersection points with a 2nd degree object

The source point $(x_s, y_s)$ is both rotated and translated while the ray direction $(x_d, y_d)$ is only rotated. The value for $\lambda$ is then found after substituting the new $x'$ and $y'$ into the object's equation. This results in a quadratic equation in $\lambda$.

$$a\lambda^2 + b\lambda + c = 0 \tag{5}$$

Solving the above equation gives us two possible quadratic roots.

In the case where $b^2 - 4ac < 0$, there is no intersection between the light ray and the object. This case is caught and returned as no intercept. The value of $a$ is required to always be non-zero while finding the intersection point(s) between the object and the light ray. This has been summarized in the illustration below -

**3rd degree** We can compute the point of intersection between the light ray and the third degree object in a similar manner as the second degree object, rotating and translating $x_s$ and $y_s$ values as well as rotating $x_d$ and $y_d$ before substituting the resulting $x'$ and $y'$ into the third degree object equation in order to find $\lambda$.

$$a\lambda^3 + b\lambda^2 + c\lambda + d = 0 \tag{6}$$

To solve the cubic equation, Eq. (6), we use *Cardano's formula*. This has been summarized in the illustration below -

$$a\lambda^3 + b\lambda^2 + c\lambda + d = 0$$

$$a_3\lambda^3 + a_2\lambda^2 + a_1\lambda + a_0 = 0$$

Cardano's equations

$$\mathcal{Q} \equiv \frac{3a_1 - a_2^2}{9}$$

$$\mathcal{R} \equiv \frac{9a_2a_1 - 27a_0 - 2a_2^3}{54}$$

$$\mathcal{D} \equiv \mathcal{Q}^3 + \mathcal{R}^2$$

$$\mathcal{S} \equiv \sqrt[3]{\mathcal{R} + \sqrt{\mathcal{D}}}$$

$$\mathcal{T} \equiv \sqrt[3]{\mathcal{R} - \sqrt{\mathcal{D}}}$$

$$\lambda_1 = -\frac{1}{3}a_2 + (\mathcal{S} + \mathcal{T})$$

$$\lambda_2 = -\frac{1}{3}a_2 - \frac{1}{2}(\mathcal{S} + \mathcal{T}) + \frac{1}{2}i\sqrt{3}(\mathcal{S} - \mathcal{T})$$

$$\lambda_3 = -\frac{1}{3}a_2 - \frac{1}{2}(\mathcal{S} + \mathcal{T}) - \frac{1}{2}i\sqrt{3}(\mathcal{S} - \mathcal{T})$$

| Discriminant | Roots |
|---|---|
| $\mathcal{D} > 0$ | 1 real, 2 complex |
| $\mathcal{D} = 0$ | 3 real, at least 2 are equal |
| $\mathcal{D} < 0$ | 3 real, all unequal |

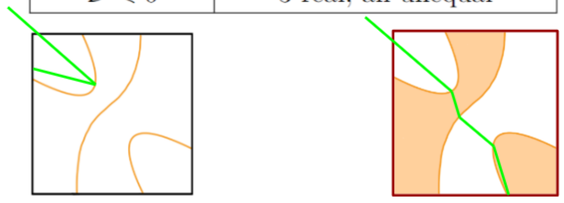


Figure 4: The Light ray vector intersection points with a $3^{rd}$ degree object

### 4.3.4 Reflection and Refraction

**Reflection** When the incident light ray intersects with a polynomial curve, it gets reflected with respect to the normal such that the angle of incidence equals the angle of reflection at the hit point on the curve.
For calculating this, we currently use the Mirroring Technique, wherein the tangent at the hit-point to the curve is treated as the mirror and the reflected light ray is found by drawing the vector between a mirrored point of the incident light ray on the opposite side of the tangent, through the hit point at the surface of the curve. This has been illustrated in Appendix Section A..

**Refraction** For refraction, the light ray bends from the rarer medium to the denser medium and vice-versa following the Snell's Law.

Refraction of light through two mediums is calculated using Snell's Law as shown below in Eq. (7).

$$n_1 sin\theta_1 = n_2 sin\theta_2 \tag{7}$$

We can also solve refraction using the Vector method shown in the Appendix Section D.

## 5 Components of the Non-optical gate system

Non-optical gates include XOR, Not-Shift as well as Permutation gates. Grugel [2023] in his Master Thesis came up with the XOR as well as Not-Shift boolean logic gates that were then incorporated into our Cryptographic system. Furthermore, we modified his original idea of Matrix-Mix gate into the Permutation gate, where we operate on the bit positions directly without storing them into matrices. We convert the original floating-point Local Bounding Box exit point coordinates $(x_e, y_e)$ of the light ray to its binary form and then apply one of these boolean gates, prioritized pseudo-randomly to it. This generates a new point in binary form which is then transformed into its floating point form to generate the projected point (Non-optical gate box position). These Non-optical gate projections take place alternatively after every interaction with an optical gate (polynomial curve) in the cryptographic scheme to make it more complex and robust against attacks. We work with boolean gates that are reversible so that symmetric nature of our crypto scheme is maintained and the encryption-decryption processes occur seamlessly.

**XOR-Gate** The XOR-Gate is a bit-wise XOR (exclusive or).

$$c[i] = a[i] \oplus b[i] \tag{8}$$

The resulting value $c$ in Eq. (8) is a number in binary form where each position is computed independently using the values $a, b$ and the XOR-truth table.

XOR is associative, commutative and each element is it's own inverse. Using these properties we show that the operation is reversible by reapplying the XOR-operation using the same value.

**Permutation** The permutation gate involves permuting (mixing up) the different bit positions of the binary form of the exit point coordinates of the Local Bounding Box, $(x_e, y_e)$. This is done, both, to the part of the (x,y)-coordinates occurring before the decimal point as well as the part after the decimal point. This allows for a new position of the projected point to be created, first in its binary form, which is then converted into the decimal (floating point) form.

**NOT-Gate followed by Left-/Right-Rotational-Shift** The NOT-Gate takes one bit-number as input: Let $a$ be an w-bit number, we compute NOT$a$ in a bit-wise manner. Let $w$ stand for *wordlength*,

$$c[i] = \neg a[i] \text{ where } 0 \leq i \leq w-1 \tag{9}$$

The resulting value $c$ in Eq. (9) is an w-bit number, where each position is computed independently using the values $a$ and the NOT-truth table. For the Left-/Right-Rotational-Shift operation after the NOT gate operation, we have a bit-number $\neg a$ and an integer $k$ as input. Depending on the direction (left/right) we move all the bits of the bit-number $\neg a$ by $k$ positions into the direction. Let $\neg a$ be a $w$-bit number, let $k$ be a integer. We take $k \mod w$, let the direction be left:

$$c[i] = \neg a[(i+k) \mod w] \text{ where } 0 \leq i \leq w-1 \tag{10}$$

Also, in the reverse direction,

$$\neg\neg a[i] = \neg c[i] = a[i] \text{ where } 0 \leq i \leq w-1 \tag{11}$$

The resulting value $c$ is an $w$-bit number, where each position is computed independently using the values $a$ and the NOT-truth table.

For the Left-/Right-Rotational-Shift operation after the NOT gate operation, we have a bit-number $\neg a$ and an integer $k$ as input. Depending on the direction (left/right) we move all the bits of the bit-number $\neg a$ by $k$ positions into the direction. Let $\neg a$ be a $w$-bit number, let $k$ be a integer. We take $k \mod w$, let the direction be left:

$$c[i] = \neg a[(i+k) \mod w] \text{ where } 0 \leq i \leq w-1 \tag{12}$$

To reverse the shift operation we apply the same number of steps in the opposite direction. So for the above example, the direction would be taken as right.

## 6 The Cryptographic scheme

### 6.1 Plaintext

The Plaintext of the scheme consists of the initial $(x_s, y_s)$ coordinate as well as the initial direction $(dx_i, dy_i)$.

### 6.2 Ciphertext

The Ciphertext of the scheme consists of the final coordinate $(x_t, y_t)$, final direction $(dx_t, dy_t)$ as well as the stack of Non-optical gate boxes.

### 6.3 Key

The Key in our cryptographic scheme consists overall of three parts and is created dynamically from the terminal during runtime. The three parts of the key include, the parameters for the Global Bounding Box, the parameters for the cryptographic element and the list of parameters for the Local Object Boxes and the object they contain within them.

The parameters for the Global Bounding Box include its point of origin, which is the $(x, y)$ coordinate of the bottom-left point, as well as the width and height of the Global Bounding Box.

The parameters for the cryptographic element include one integer as the seed for the random number generator within the cryptographic element and one floating point number which is the width of the Global Bounding box. The seed is specified by the user during runtime and fixes the number, type and positions of the object boxes during that particular execution so that we can perform encryption and decryption over the same setup.

This is followed by the list of parameters for the object boxes which include the type of object ($2^{nd}$ degree, $3^{rd}$ degree etc.), the type of Local Object box (reflection/refraction), $(x, y)$ coordinates of the bottom-left of the object box, it's width and height, the object contained within, including the different coefficients of the terms associated with the first, second or third degree polynomial (e.g: coefficients of $x^2$, $y^2$ etc.), the refractive indices of the respective media (if it is a 'refraction' Local Object box), one 8 bit string used as the mask for the part of the XOR number before the decimal place and another 512 bit string for the part of the XOR number after the decimal place, followed by mappings for the permutation gate as well as reverse of the permutation gate. We have the unique identifier which is used to link a Non-Optical gate box to the correct Local Object Box. Finally, the key contains two integers : the first is the amount of shift positions and the second is the shift direction for the Shift-Gate. 1 signifies a right shift and 0 signifies a left shift in our scheme.

The components of a sample key (for the same instance as shown in Figure 2 ) has been illustrated in the Appendix E.

## 6.4 Encryption

The encryption step begins when the light ray enters the Global Bounding Box at a certain initial $(x_s, y_s)$ coordinate as well as the initial direction $(dx_i, dy_i)$ (Plaintext). The light ray then interacts with the first Local Bounding Box in its path. If the light ray is in direct line of contact with the curve object inside the Local Bounding Box, then it behaves in either of the two ways, i.e; if the Local Bounding Box happens to be Reflective, then the light ray reflects from the surface of the polynomial curve, following the Law of Reflection (the incident angle equals the reflection angle, at the point of contact, w.r.t the normal at that point) and if the Local Bounding Box happens to be Refractive, then the light ray interacts with the polynomial curve on the basis of Snell's law of reflection. Once this step is completed, the light ray then comes into contact with the Local Bounding Box boundary. At this exit point of the Local Box Boundary, $(x_e, y_e)$, this floating point coordinate is transformed into it's binary form and one of the three Non-optical (boolean) gates are pseudo-randomly selected (assigned the priority of 0,1 or 2) and then applied to this binary form of the (x,y)-coordinate. This results in another (x,y)-coordinate in binary form, which is then transformed back into floating point form in order to obtain the projected point, or in other words, the creation of the Non-optical gate box position. The light ray then continues from this newly created Non-optical gate box position, retaining the same direction that it had while exiting the previous Local Bounding Box until it touches another polynomial curve inside a different Local Bounding Box. This alternating interaction between polynomial curves as well as projection to Non-optical Gate boxes continues until the light ray path terminates at the boundary of the Global Bounding Box and we obtain the final $(x_t, y_t)$ coordinate as well as the final direction $(dx_t, dy_t)$ of the light ray, along with the stack of Non-Optical gate boxes (Ciphertext). An example encryption on a sample key has been shown in Figure 2.

### 6.4.1 Non-Optical Gate Box

The Non-Optical gate box does not contain any object and is more important to be considered during the decryption process (except, that in the encryption process no two Non-optical gate boxes can overlap). After a successful manipulation of a point $P = (x_e, y_e)$ using one of the boolean logic gates discussed in Section 5, to a point $P' = (x', y')$, we create a Non-Optical gate box at the position $P'$.

This Non-Optical gate box contains the point $P'$, a boundary, an unique identifier to the object box, the type of boolean logic-gate that has been applied, and the number of places to the right of the decimal point of $(x', y')$. This is important for the process of decryption, since the $P' = (x', y')$ should get mapped to it's correct binary form.

During the encryption process, Non-Optical gate boxes are pushed to the stack within the 2D environment. During the decryption process, Non-Optical gate boxes are popped from the stack within the 2D environment, which helps retrace the path of the light ray backwards to the initial position of it's entry into the Global Bounding Box, at $(x_s, y_s)$.

### 6.4.2 Validity of Non-Optical Gate Positions

We find the conditions for which the projection of a point $P$ to $P'$ from a Local object box to a certain point in the 2D environment can be carried out.

We consider a point $P' = (x', y')$ as invalid if it is outside of the Global Bounding Box, within the same Local Object Box, within another Non-Optical gate Box or potentially enclosed in a closed object such an ellipse or a polygon formed by intersection of linear equations.

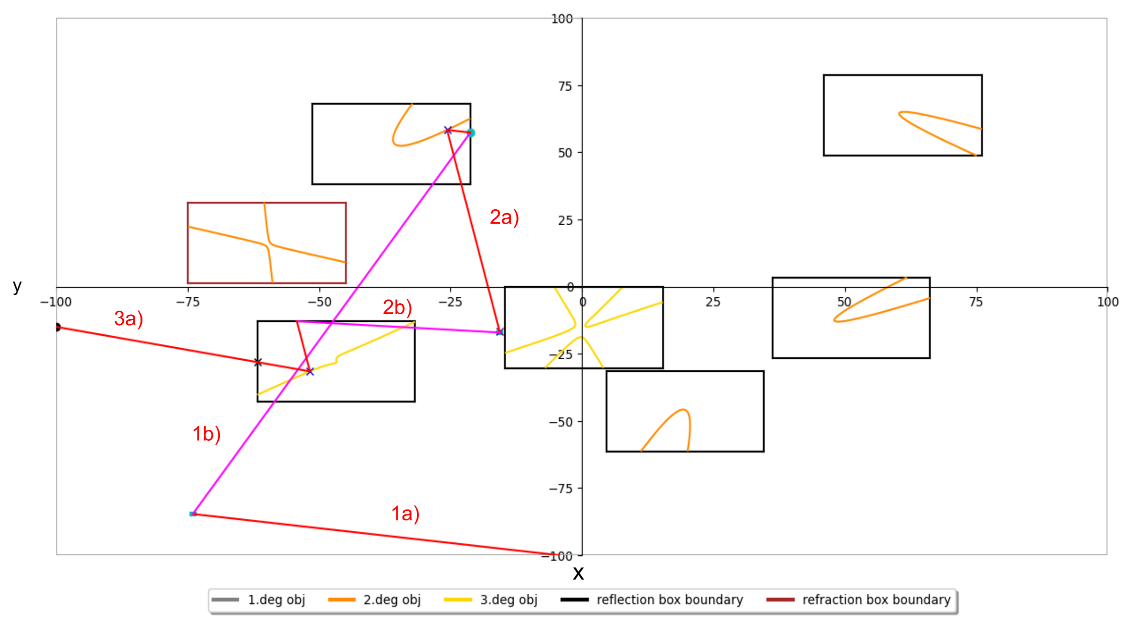


Figure 5: Decryption on sample key presented in Figure 2

If any of the following conditions are fulfilled, then the point $P'$ is outside of the Global Bounding Box $G = (x_G, y_G, \text{width}_G, \text{height}_G)$ and is therefore invalid : $x' < x_G, x' > x_G + \text{width}_G, y' < y_G, y' > y_G + \text{height}_G$.

To verify if the point $P'$ is within the same Local object Box or within another Non-optical gate Box, we use similar conditions like above but instead of testing for outside of the box, we test for inside of the box. We achieve this by replacing every $<$ with $>$ and vice-versa.

The next case we need to verify is whether $P'$ is enclosed by an object. For objects of the $1^{st}$ degree, we test whether the point $P'$ is contained within it's four vertices. For $2^{nd}$ degree objects we need to consider the case where the $2^{nd}$ degree object is an ellipse. For this, we compute the value of the formula of the ellipse with the point $P'$ as input. For the two aforementioned types of objects, if the point $P'$ is within the object then we consider the position as invalid.

If none of the conditions discussed above, which lead to an invalid point is fulfilled, we consider the point $P'$ as a valid position, and therefore the projection from $P$ to $P'$ is applied.

### 6.5 Decryption

The decryption process involves the exact opposite process as the encryption, with the process starting at the final $(x_t, y_t)$ coordinate as well as the final direction $(dx_t, dy_t)$ of the Global Bounding Box and ending at the Plaintext, $(x_s, y_s)$ coordinate as well as the initial direction $(dx_s, dy_s)$. A sample decryption for the same key and encryption instance as in Figure 2 has been shown below in Figure 5.

## 7 Attacks

We also attack our cryptographic system in order to test its robustness. We find that some parts of the cryptographic system could sometimes be broken for bad keys. We have widely classified the attack system into two parts, i.e; attack on the entire 2D setup (X-ray attack or Overall Attack), in order to attempt to locate all the Local Bounding Boxes, followed by the Box-by-Box attacks in order to try and 'discover' the objects (curves) inside each of the Local Bounding Boxes as well as the bit-by-bit reconstruction method used to estimate the Boolean gate applied per Local Bounding Box.

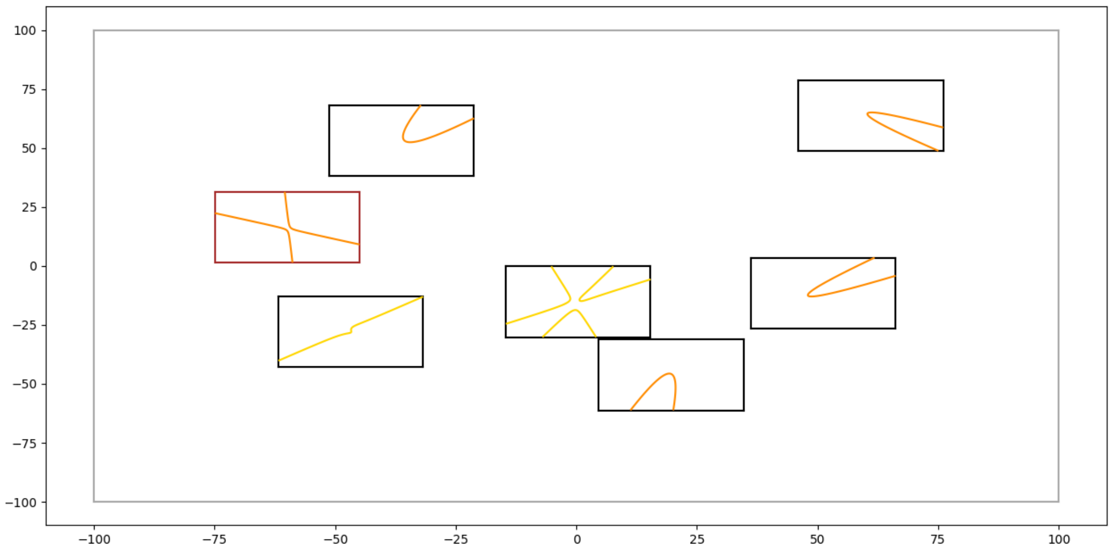


Figure 6: Clustering output on sample Key

## 7.1 X-Ray Attack (Overall Attack)

The overall attack is launched on the Global Bounding Box to locate the smaller Local Bounding boxes using a grid-attack followed by clustering. In the grid-attack technique, the Global bounding box is first hit with multiple vertical and horizontal rays and the points from which the rays get deflected could be potential locations of the Local Bounding Boxes.

### 7.1.1 Clustering

If the encrypted data contains the ground-truth point of the applied non-optical gate, then they can be sorted by the box-id of their respective box of origin. However, if the attacker does not have access to this data, which is true in our case, then the reconstruction of the Local Bounding Boxes becomes even more difficult, especially when the there aren't enough light rays hitting certain Local Bounding Boxes and therefore not enough samples for certain Local Bounding boxes, when the light ray is initialized at the boundary of the Global Bounding Box. However, if for each box, enough such samples are obtained, the minimum and maximum of the x and y coordinates can be potentially computed. These could give at least an approximation of the actual boundary. If the boxes' size is known prior to the attack, fewer samples are needed, as points on three different sides are sufficient to reconstruct the box correctly. However, we also don't give out this information about our cryptographic system to the attacker, ensuring the difficulty in locating the Local Bounding Boxes in a given 2D setup.

The clustering output for the key shown in Figure 2 has been illustrated here in Figure 6. As can be noticed, the Local Bounding Boxes could not be 'discovered' in this case, using this method.

The pseudo-code for the clustering with known ground truth points is provided in the Appendix Section G. In this, we assume that we also give away information about the size of the boxes to the attacker.

## 7.2 Box-by-box attacks

In the Box-by-box attacks, we attack the polynomial curves (objects) as well as Non-optical boolean gates associated with the individual Local Bounding Boxes, assuming that we made the discovery of the position and sizes of the Local Bounding Boxes inside the Global Bounding Box. For the box-by-box attacks, the data points on the individual polynomial curves are first attempted to be found using Binary Search and then followed by approximating the curves using various techniques using those data points.

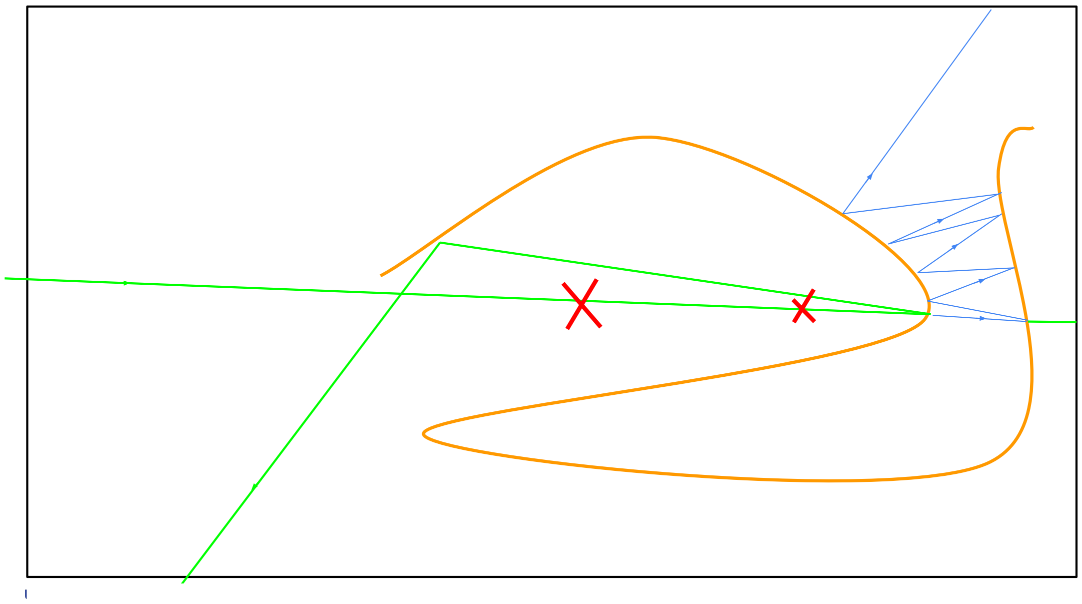


Figure 7: Binary Search

### 7.2.1 Binary Search

The Binary Search approach iteratively shrinks the interval along a light-ray, comparing the current exit point with the exit point of the original light-ray whilst having the same starting direction. Robens [2024], during his Bachelor Thesis came up with this intuitive but extensive technique to try and locate data-points on a polynomial curve (object).

In this technique, the interval converges at the position of the first interaction with an optical gate along the trajectory of the light-ray. This can be seen in the Figure 7. The dark green light-ray is the initial light-ray along which the subsequent light-rays (marked by the red crosses) are set.

This method is sometimes able to discover simple curves, but it takes a long run-time, since the system must simulate many light rays during the search process, and each light ray yields only one data point on the surface of the polynomial curve. Part of the computation can be reused, as it is likely that some samples for the reconstruction of the non-optical gates can be gathered during the process. The problem with this is that only a limited amount of different options are explored, since about half of the light-rays simulated for each data point obtained share the same trajectory and therefore yield only duplicate samples for the reconstruction.

The pseudo-code for the Binary Search is provided in the Appendix Section G.

### 7.2.2 Approximation Techniques

**Linear Objects** For approximating linear objects, we use the property that at least two data points are needed in order to draw a line through it. However, since our linear objects consist of multiple sides and we don't know the corresponding side of each data point, we use at least three data points per side to confirm that they belong to that particular side of the polygon/open-sided linear object.

We start by gathering all data point pairs and their corresponding line equations. We narrow this search down to the line equations that cover more than two data points. All the remaining data point pairs then get assigned to their corresponding line equations. To find the corners, we first find the most distant data point from each of the other line equations to achieve better precision when finding the corners. We then calculate the intersection of all valid lines. By solving these, we retrieve a set of possible corner point coordinates which include the actual corners and false or redundant corners. Since we only allow simple polygons (with the sides not crossing over each other), we can group these lines as adjacent and opposite ones. The closer two opposing lines are to being parallel, the further away their intersection, in this case the more likely it will be a redundant corner. Therefore in some cases redundant corners can be discarded by introducing a bound. However, this is not trivial since it is not clear how large the bound could be, as parts of the polygon are allowed to be outside the Local Bounding Box. Also, sometimes there is an open side to the polygon, we will not be able to find data points on the open side, since there can not be data points on the missing edge. Therefore we order the already found 'actual' corners in a chain to determine where the open side is. Sometimes, if one or more of the sides of the polygon does not get hit by sufficient number of light rays, because of its close proximity to another Local Bounding Box in the 2D setup, then we miss out on computing these edges of the polygon altogether.

The algorithm to approximate linear objects is given in the Appendix Section G.

**Non-Linear Objects** The reconstruction of Non-Linear objects is first attempted to be done via the Naive Approaches and then replaced by the Approximation techniques involving PLU factorization of the Jacobian of Non-linear equations and thereafter using the Modified-Newton (md-newton) step. We experimented with interpolation of the data points using B-spline curve fitting. However, due to segmentation of the splines, this method was a bad fit for sparse regions of the curve where the data point density was low. This could also not be used to segment discontinuous curves.

We also could not perceive of a way to segment non-continuous curves, especially when overlapping. We thereafter used some other non-linear Python modules such as Gekko and Scipy.Optimize to approximate the curves but without sufficiently good enough results. Then, we moved onto Global Simulated Annealing to find the curve coefficients, offset and rotation and possibly decide what type of object it is. This partially worked in finding local minima but failed for overall curve approximations.

For a system of Non-linear equations, we first compute the Jacobian matrix, J of the system, use PLU decomposition of the Jacobian to solve the linearized system and finally use the modified Newton method iteratively until convergence. As part of his Bachelor Thesis, Leisegang [2024] experimented with this method. This worked in some cases, where we were able to approximate certain parts of the individual polynomial curves (objects), given sufficient amount of data points. But it also failed in some cases, especially while estimating $3^{rd}$ degree curves. This has been covered in the Appendix SectionG in more detail.

### 7.2.3 Deciding the object type

Once the curve approximations are assumed to be done, the next step is to assign the polynomials to $1^{st}$, $2^{nd}$ or $3^{rd}$ degree type. Deciding if a set of data points belongs to a linear object is easier to do than deciding if they belong to higher degree curves (objects) especially when these data points are spatially separated from each other within a small $\epsilon$.

Difficulties of deciding whether the polynomial curve is of $2^{nd}$ or of $3^{rd}$ degree type on the basis of the residuals are as follows:
When finding a unique solution for a baseline $3^{rd}$ degree function, the objective function of $2^{nd}$ degree, due to it's lower cardinality of variables and therefore more sparse information about the polynomial curve (object), may satisfy the linear equation system better than the objective function of $3^{rd}$ degree.
Similarly, a $2^{nd}$ degree object may satisfy a $3^{rd}$ degree objective function, because a generic $3^{rd}$ degree polynomial includes all terms of a generic $2^{nd}$ degree polynomial. However, this leads to arbitrary curves.

To mitigate this issue, we tried choosing suitable data points on the polynomial curve to construct a linear equation system, that captured less localized properties of the curve. We selected data points with cardinality 6 / 10 with the maximum Manhattan distance between points out of the set of all data points. This helped somewhat with the reduction of both, very local as well as very sparse data points.

The Algorithm for deciding the object type is shown the Appendix Section G.

### 7.2.4 Bit-by-bit reconstruction

Once the polynomial curves (objects) have been discovered and assigned to their respective types, the next step is to estimate the Non-optical boolean gates assigned to each of the Local Bounding boxes. We assume here that the ground-truth points at the exit of every Local Bounding Box is known for every light ray interacting with such a Local Bounding Box. These exit points are however, not shared in reality by us to the attacker who interacts with our cryptographic system.

Since the encryption gate is chosen pseudo-randomly and an integer out of 0,1 or 2 is returned to indicate which non-optical gate is used, for the particular Local Bounding Box, one needs to check for all combinations, since for a box, 0 might refer to permutation, 1 might refer to Not-Shift but for the next Local Bounding Box, 0 might refer to XOR and so on. To determine which gate was actually used according to encryption priority, the reconstruction algorithm should therefore compute a measure of certainty or correctness of output to choose the correct transformation for each of the priorities. For this task, in the case of the XOR and the Not-Shift, the average precision of the computed transformation is computed as $\Sigma \frac{precisionperbit}{numberofbits}$ with precision per bit = $\frac{numberofsamplessupportingbit}{numberofsamples}$ . For all possible combinations of priorities to encryption methods, the sum of the average precision of the reconstructions is maximized to decide which is the most likely choice. The term, 'enough' samples needs to be addressed because it is highly subjective and depends firstly on the gate type to reconstruct and, secondly, on how the samples are created. If the samples are random, due to the light ray hitting the Local Bounding Box from random entry points, more number of samples are needed than in the case in which specific coordinates are encrypted with the functions.

**XOR** As the inverse operation of XOR is itself XOR with the same parameter ($a = (a \oplus b) \oplus b$), the secret constant $c$ can be computed by $c = t \oplus o = (o \oplus c) \oplus o$ where $t$ is the binary coordinate after the transformation and $o$ is the original binary coordinate.
For each pair of coordinates, a constant is computed. Over all of these constants, a majority gate is applied bitwise to generate a single constant. The percentage of correct bits in all previously computed constants is taken as the certainty measure to decide if XOR was the correct gate for these samples. The pseudo-code for the reconstruction of the XOR-gate is provided in the Appendix Section G. An example of the reconstruction is provided in the Appendix Section F.

**Not-Shift** To compute possible parameters for a Not-Shift operation, first $\overline{o}$ is computed by negating $o$ bitwise. Then the reconstruction iterates over all possible shifts and stores the parameters of the shifts with the best match. This process is repeated for every pair of coordinates. Since a cyclic shift is used, there exist multiple correct solutions for a shift, as for example, a right-shift by 2 and a left-shift by -2 achieve the same transformation. If the concrete implementation is known, this can be further optimized to only check the shifting amounts in the range that can actually get generated. The measure of certainty in the reconstruction is the determined as the percentage of bits correctly set in each sample according to the shift that yields the most correct matching.

Without the need to decide if Not-Shift was the actually applied gate, it is possible to reconstruct the correct parameters with less number of samples, if any shift can be uniquely detected (not accounting for shifts further than the length of the coordinate). A possible candidate for this would be a coordinate that contains only a single 1 in its binary representation at a specific place in the decimal part and everywhere else 0. Using random samples without duplicates, the number of required samples is much higher. The pseudo-code for the reconstruction of the Not-Shift-gate is provided in the Appendix Section G. An example of the reconstruction is provided in the Appendix Section F.

**Permutation** The reconstruction of the Permutation gate is the computationally most expensive reconstruction operation out of the three cases. Since every bit-position has to be reconstructed, a lot of samples are required, especially when random samples are used instead of specifically chosen entry points of the light into the Local Bounding Box $(x_e, y_e)$, hence leading to specific exit points at the boundary of the Local Bounding Box. The number of samples required for reconstruction significantly increases with the number of bits we use during encryption (currently, 512). For each coordinate pair, the reconstruction computes which bit-index in the transformed coordinate could stem from which indices in the original coordinate. Obtaining the correct mapping can be performed by set-intersection for sets of original indices for each bit-position in the transformed coordinate. This procedure checks for the correct transformation, as the correct transformation could be a possibility in every sample.

The certainty measure in this case is first computed bit-wise on the resulting sets. It is calculated by the inverse of the number of elements per set, e.g. if there remain two possibilities for a given index, the certainty is $\frac{1}{2}$. If there is the special case that there is an empty set, it is impossible that the transformation could have been a Permutation, as the correct permutation is always preserved during set-intersection. In this case, the reconstruction of the gate can be stopped, and the certainty is 0. Otherwise, the certainty is taken to be the average of the bit-wise certainties.

Assuming specific coordinates can be chosen to be encrypted, the amount of needed samples varies strongly between possible approaches. The naive approach for chosen coordinates is to take coordinates containing either exactly one 0 or exactly one 1 and repeat this for every bit position. A more optimized version attempts to reconstruct the Permutation gate to an extent with a logarithmic number of samples relative to the length of the input. For random sampling, more coordinate pairs are required. For the more refined algorithm, it is needed that the coordinates getting encrypted can be freely chosen. As this requirement is not fulfilled, the less efficient algorithm is used. The pseudo-code for the reconstruction of the Permutation gate is provided in the Appendix Section G. An example of the reconstruction is provided in the Appendix Section F.

## 8 Results of Box-by-Box Attacks

In the Figure 8 is one among the several boxes of the actual (left) vs computed (right) box-by-box outputs belonging to the sample Key shown in Figure 2. As can be seen, the curve has been wrongly estimated. These errors could also occur while estimating the Non-optical gate (boolean gate) associated with such boxes.

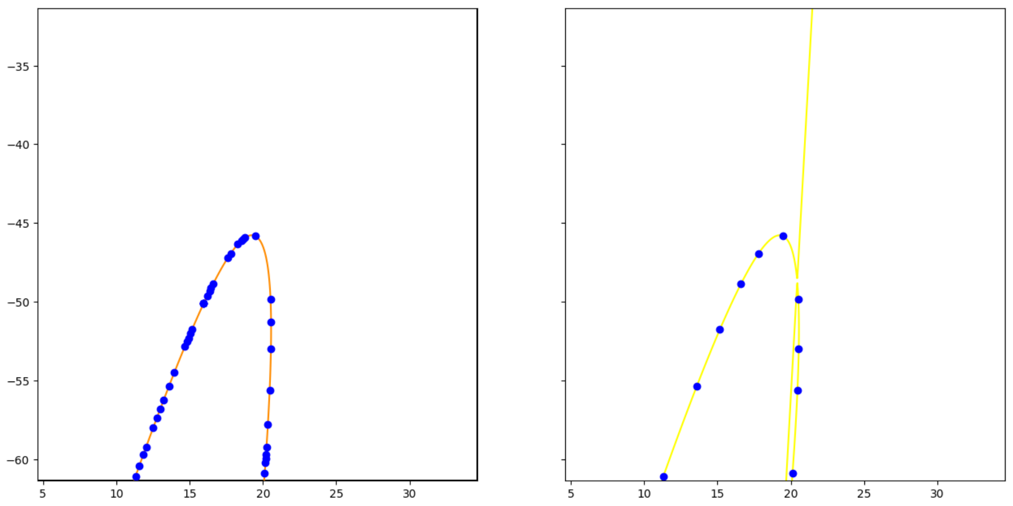


Figure 8: Mismatch in the actual vs the estimated curve from box-by-box attack

## 9 Conclusion and Future Work

We have been able to introduce a completely novel symmetric key cryptographic system in 2D using a light ray's alternating interaction between reflective as well as refractive polynomial curves (objects) of $1^{st}$, $2^{nd}$ and $3^{rd}$ degree type along with boolean gates, such as; XOR, NOT-Shift and Permutation. This two-fold interaction and projection of the light ray involving the various objects in our 2D setup creates a robust and secure cryptographic system which requires very specific types of attacks to even partially investigate it. The key generation step ensures that our keys have the optimal local bounding box to global bounding box size ratio as well as the number of Local Bounding Boxes are numerous enough to generate safe keys. By incorporating higher number of bits (currently, 512) into our cryptographic system, we make sure that there is no loss in bits related data during the encryption-decryption process. We plan to further delve into the Precision Analysis aspect of the cryptographic system in a pure mathematical sense, as a Future Work. We would also like to create and analyze the same cryptographic system, but in 3D.

## References

John H. Reif, J. Doug Tygar, and A. Yoshida. Computability and complexity of ray tracing. *Discrete & Computational Geometry*, 11:265–288, 1994.

JongWook Han, Choon-Sik Park, Dae-Hyun Ryu, and Eun-Soo Kim. Optical image encryption based on XOR operations. *Optical Engineering*, 38(1):47 – 54, 1999. doi:10.1117/1.602060. URL `https://doi.org/10.1117/1.602060`.

Ethan L Blansett, Richard Crabtree Schroeppel, Jason D Tang, Perry J Robertson, Gregory Allen Vawter, Thomas David Tarman, and Lyndon George Pierson. Photonic encryption using all optical logic. 12 2003. doi:10.2172/918388. URL `https://www.osti.gov/biblio/918388`.

George W. Tokarsky. Polygonal rooms not illuminable from every point. *The American Mathematical Monthly*, 102 (10):867–879, 1995. ISSN 00029890, 19300972. URL `http://www.jstor.org/stable/2975263`.

Tobias Grugel. Implementation, simulation and analysis of a gate-based cryptographic scheme in a ray tracing environment. Unpublished Master Thesis, June 2023.

Benjamin Robens. Analysis and defense against attacks on visual cryptographic schemes with optical and non-optical gates. Unpublished Bachelor Thesis, May 2024.

Maximilian Leisegang. Visual cryptography: Attacks on visual cryptographic system. Unpublished Bachelor Thesis, July 2024.

## A Mirror Technique for finding the Reflected Ray

The mirroring technique uses the tangent as the mirror at the point of intersection of the light ray with the object. The point mirrored by the tangent is determined using the mid-point theorem of a line segment. On tracing a line through this mirrored point and the point of intersection, we obtain the reflected light ray.

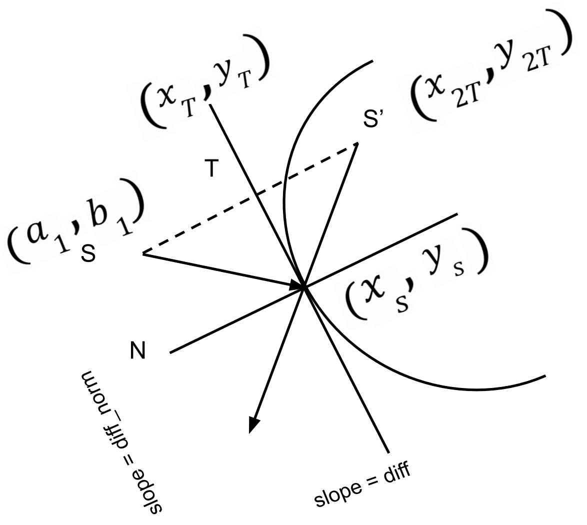


Figure 9: The Mirroring Technique

$$y_T = x_T.\text{diff} + c_1 \tag{13}$$

$$y_T = (x_T - a_1).\text{diff\_norm} + b_1 \tag{14}$$

By rearranging Eqs.(13) and (14), we obtain :

$$x_T = \frac{c_1 - b_1 + a_1.\text{diff\_norm}}{\text{diff\_norm} - \text{diff}} \tag{15}$$

Using the mid-point theorem of a line segment,

$$x_T = \frac{x_{2T} + a_1}{2} \tag{16}$$

$$y_T = \frac{y_{2T} + b_1}{2} \tag{17}$$

Rearranging (16) and (17), we obtain :

$$x_{2T} = 2x_T - a_1, y_{2T} = 2y_T - b_1 \tag{18}$$

Using (15) and (13), we obtain $(x_{2T}, y_{2T})$.

We also know that,

$$\frac{y - y_{2T}}{x - x_{2T}} = \frac{y_s - y_{2T}}{x_s - x_{2T}} \tag{19}$$

This gives us

$$y = \frac{(y_s - y_{2T})(x - x_{2T})}{x_s - x_{2T}} + y_{2T} \tag{20}$$

Solving this results in the equation of the reflected light ray (as the equation of a line).

## B Method using $tan\theta$ for finding the Reflected Ray

This method uses the fact that the angle between the incident ray and the normal to the surface of the object is the same as the angle between the reflected ray and the normal.

Assume that the slope of the incident light ray is $m_1$ and the slope of the normal is $m$. Let the slope of the reflected light ray be $m_2$. We know that as per the Law of Reflection, the angle between the incident ray and the normal is the same as the angle between the normal and the reflected light ray.

We hence come up with the following Formula :

$$\frac{m_1 - m}{1 + m_1 m} = \frac{m - m_2}{1 + m_2 m} \tag{21}$$

Solving this would give us a quadratic equation which would in turn result in two possible values for the slope of the reflected light ray, $m_2$.

## C Vector Technique for finding the Reflected Ray

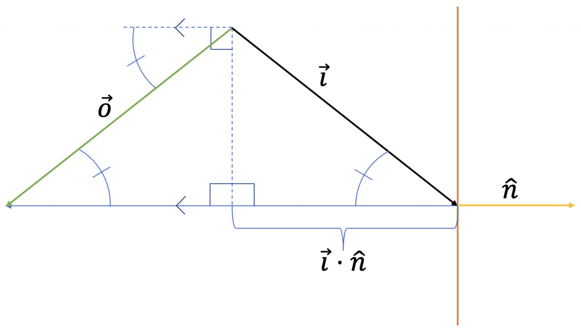


Figure 10: Vector Technique for finding the Reflected Ray

To calculate the reflected ray using vectors, we note that the inputs are the input ray, $\vec{i}$, and the normalized normal vector into the object at the point of intercept, $\hat{n}$, which by definition has a magnitude value of one. As seen in the diagram, the dot product of these two vectors, $\vec{i} \cdot \hat{n}$, is the projection of the input ray onto $\hat{n}$. Since this is a scalar, multiplying it by $\hat{n}$ changes the magnitude of $\hat{n}$ but not its direction. By then multiplying this value by two and subtracting it from $\vec{i}$, we obtain an output vector $\vec{o}$. The vector $\vec{o}$ has the same direction as the output ray due to reflection. We can see this is true, due to the fact that reflection arises from the angle between the incidence ray and the normal being equal to the angle between the normal and the output ray. The vectors here create two right triangles, one with sides $\|\vec{i}\|$, $\vec{i} \cdot \hat{n}$, and $\|\vec{i} - (\vec{i} \cdot \hat{n})\hat{n}\|$; the other with sides $\|\vec{o}\|$, $\vec{i} \cdot \hat{n}$, and $\|\vec{i} - (\vec{i} \cdot \hat{n})\hat{n}\|$. These triangles are equivalent, due to two sides and the angle between them being equivalent. Thus, the angle between $\vec{o}$ and $\hat{n}$ is equal to the angle between $\vec{i}$ and $\hat{n}$. Therefore, $\vec{o}$ is the output ray from reflection.

## D Vector Technique for finding the Refracted Ray

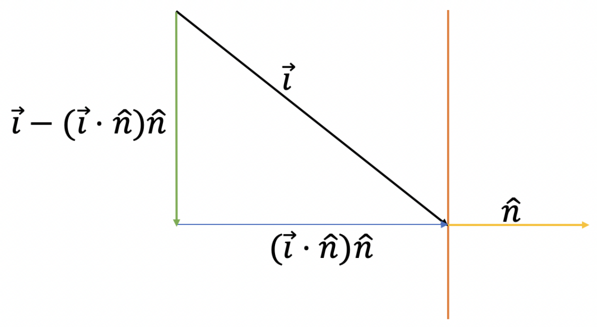

$\vec{\imath}$
$\vec{\imath} - (\vec{\imath} \cdot \hat{n})\hat{n}$
$\hat{n}$
$(\vec{\imath} \cdot \hat{n})\hat{n}$

Figure 11: Vector $\vec{i}$ derived from addition of orthogonal vectors

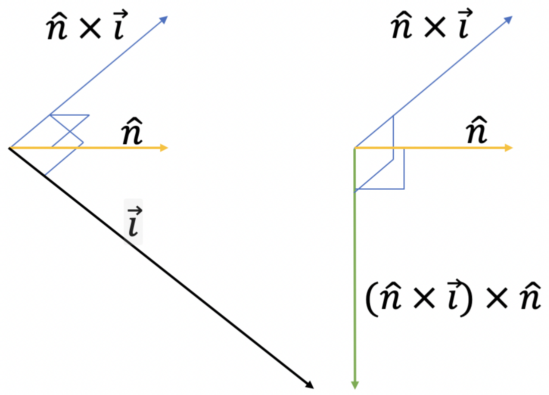

$\hat{n} \times \vec{\imath}$
$\hat{n}$
$\vec{\imath}$
$\hat{n} \times \vec{\imath}$
$\hat{n}$
$(\hat{n} \times \vec{\imath}) \times \hat{n}$

Figure 12: Use of cross product of vectors in finding Refracted ray

Refraction using vectors follows from the same initial point as reflection using vectors. We note from Figure 11 that $(\vec{i} \cdot \hat{n})\hat{n}$ and $\vec{i} - (\vec{i} \cdot \hat{n})\hat{n}$ are two orthogonal vectors that, when added together, produce vector $\vec{i}$. There is a second method to calculate the same direction as $\vec{i} - (\vec{i} \cdot \hat{n})\hat{n}$ using the cross product, $(\hat{n} \times \vec{i}) \times \hat{n}$ as seen in the Figure 12.

Also, in the Figure 12, note that all right angles are denoted. The value for $\hat{n} \times \vec{i}$ points in a third direction, orthogonal to both $\hat{n}$ and $\vec{i}$. Meanwhile, $(\hat{n} \times \vec{i}) \times \hat{n}$ is in the same plane as $\hat{n}$ and $\vec{i}$. With this equivalence, we can say,

$$\vec{i} = (\vec{i} \cdot \hat{n})\hat{n} + \vec{i} - (\vec{i} \cdot \hat{n})\hat{n} = (\vec{i} \cdot \hat{n})\hat{n} + (\hat{n} \times \vec{i}) \times \hat{n} \tag{22}$$

For simplicity, we normalize $\vec{i}$ as $\hat{i}$. Snell's Law is the equation to calculate refraction from one medium into another using

$$n_1 sin\theta_1 = n_2 sin\theta_2 \tag{23}$$

Setting $\mu = n_1/n_2$, this can be rewritten as

$$\mu sin\theta_1 = sin\theta_2 \tag{24}$$

The definition of cross product, for two vectors A and B with angle $\theta$ between them, states

$$sin\theta = (A \times B)/\|A\|\|B\| \tag{25}$$

We can now rewrite Snell's Law for some output vector $\vec{r}$ using the cross products as

$$\mu(\hat{n} \times \hat{i}) = \hat{n} \times \vec{r} \tag{26}$$

The vector $\vec{r}$, like $\hat{i}$, can be expressed as

$$\vec{r} = (\vec{r} \cdot \hat{n})\hat{n} + \vec{r} - (\vec{r} \cdot \hat{n})\hat{n} = (\vec{r} \cdot \hat{n})\hat{n} + (\hat{n} \times \vec{r}) \times \hat{n} \tag{27}$$

Using this, we can substitute the first cross product with Snell's Law:

$$\vec{r} = (\vec{r} \cdot \hat{n})\hat{n} + \mu(\hat{n} \times \hat{i}) \times \hat{n} \tag{28}$$

The cross products can be replaced once again, giving us

$$\vec{r} = (\vec{r} \cdot \hat{n})\hat{n} + \mu(\hat{i} - (\hat{i} \cdot \hat{n})\hat{n}) \tag{29}$$

Now, $\vec{r}$ must be removed from the right side of the equation. This can be done by setting $\vec{r}$ to a normalized vector, $\hat{r}$.

$$\hat{r}^2 = ((\vec{r} \cdot \hat{n})\hat{n})^2 + (\mu(\hat{i} - (\hat{i} \cdot \hat{n})\hat{n}))^2 + 2(((\vec{r} \cdot \hat{n})\hat{n}) \cdot (\mu(\hat{i} - (\hat{i} \cdot \hat{n})\hat{n}))) \tag{30}$$

As the two components are orthogonal, the dot product between them is zero.

$$\hat{r}^2 = (\vec{r} \cdot \hat{n})^2\hat{n}^2 + \mu^2(\hat{i}^2 - 2(((\hat{i} \cdot \hat{n})\hat{n}) \cdot (\hat{i})) + (\hat{i} \cdot \hat{n})^2\hat{n}^2) \tag{31}$$

Since $\hat{n}$ and $\hat{i}$ are normalized vectors, their squares are simply one.

$$\hat{r}^2 = (\vec{r} \cdot \hat{n})^2 + \mu^2(1 - 2(\hat{i} \cdot \hat{n})2^+(\hat{i} \cdot \hat{n})^2) = (\vec{r} \cdot \hat{n})^2 + \mu^2(1 - (\hat{i} \cdot \hat{n})^2) \tag{32}$$

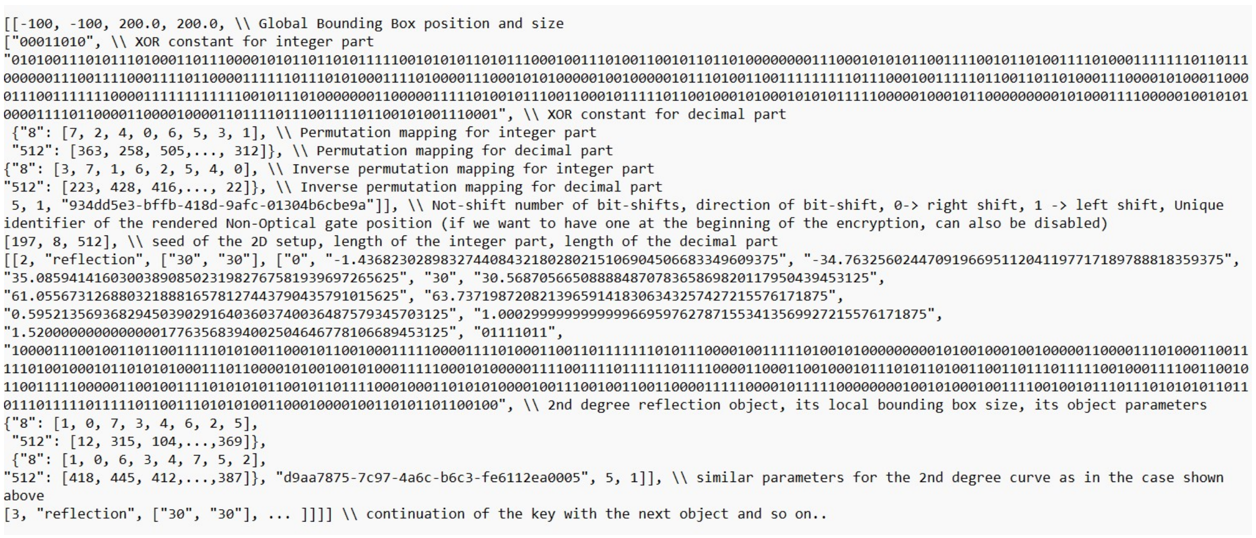
Finally, since $\hat{r}^2 = 1$ as well,

$$(\vec{r} \cdot \hat{n}) = \pm\sqrt{1 - \mu^2(1 - (\hat{i} \cdot \hat{n})^2)} \tag{33}$$

. We know the square root is positive, as the angle between $\hat{r}$ and $\hat{n}$ is always less that $\pi/2$. Thus we have the solution

$$\vec{r} = \mu(\hat{i} - (\hat{n} \cdot \hat{i})\hat{n}) + \hat{n}\sqrt{1 - \mu^2(1 - (\hat{n} \cdot \hat{i})^2)} \tag{34}$$

## E Parts of a Sample Key



```
[[-100, -100, 200.0, 200.0, \\ Global Bounding Box position and size
["00011010", \\ XOR constant for integer part
"0101001110101110100011011100001010110110101111100101010110101110001001110100110010110110100000000111000101010110011110010110100111101000111111110110111
00000011100111100011110110000111111011101010001111010000111000101010000010010000001011101001100111111111011100010011111011001101101000111000010100011000
01110011111110000111111111111001011101000000011000001111101001011100110001011111011001000101000101010111110000010001011000000000101000111100000100 10101
0000111101100001100001000011011110111001111011001010011100 01", \\ XOR constant for decimal part
 {"8": [7, 2, 4, 0, 6, 5, 3, 1], \\ Permutation mapping for integer part
 "512": [363, 258, 505,..., 312]}, \\ Permutation mapping for decimal part
{"8": [3, 7, 1, 6, 2, 5, 4, 0], \\ Inverse permutation mapping for integer part
"512": [223, 428, 416,..., 22]}, \\ Inverse permutation mapping for decimal part
 5, 1, "934dd5e3-bffb-418d-9afc-01304b6cbe9a"]], \\ Not-shift number of bit-shifts, direction of bit-shift, 0-> right shift, 1 -> left shift, Unique
identifier of the rendered Non-Optical gate position (if we want to have one at the beginning of the encryption, can also be disabled)
[197, 8, 512], \\ seed of the 2D setup, length of the integer part, length of the decimal part
[[2, "reflection", ["30", "30"], ["0", "-1.4368230289832744084321802802151069045066833496093750", "-34.7632560244709196695112041197717189788818359375",
"35.085941416030038908502319827675819396972656250", "30", "30.568705665088884870783658698201179504394531250",
"61.055673126880321888165781274437904357910156250", "63.73719872082139659141830634325742721557617187500",
"0.595213569368294503902916403603740036487579345703125", "1.000299999999999966959762787155341356992721557617187500",
"1.5200000000000000177635683940025046467781066894531250", "01111011",
"1000011100100110110011111010100110001011001000111110000111101000110011011111110101110000100111110100101000000000101001000100100000011000011101000110011
11101001000101101010100011101100001010010010100011111000101000001111001111011111101111000011000110010001011101011010011001101110111110010001111001100 10
11001111100000110010011110101010110010110111100010001101010100001001110010011001100001111100001011111000000000100101000100111100100101110111010101011011
0111011111011111011001110101010011000100001001101011011 00100", \\ 2nd degree reflection object, its local bounding box size, its object parameters
{"8": [1, 0, 7, 3, 4, 6, 2, 5],
 "512": [12, 315, 104,...,369]},
 {"8": [1, 0, 6, 3, 4, 7, 5, 2],
"512": [418, 445, 412,...,387]}, "d9aa7875-7c97-4a6c-b6c3-fe6112ea0005", 5, 1]], \\ similar parameters for the 2nd degree curve as in the case shown
above
[3, "reflection", ["30", "30"], ... ]]]] \\ continuation of the key with the next object and so on..
```

Figure 13: Components of the key, for the instance shown in Figure 2

## F Reconstruction Examples

### XOR

| Samples | (110101, 010000) | (010011, 110110) | (010111, 110010) |
|---|---|---|---|
| Intermediate | 100101 | 100101 | 100101 |
| Resulting constant | 100101 | | |
| Bitwise certainty | 1, 1, 1, 1, 1, 1 | | |

Table 2: Sample XOR reconstruction

**Not-Shift**

| Samples | (101010, 101010) | (001110, 001110) | (111101, 010000) |
|---|---|---|---|
| Possible right shifts | 1, 3, 5 | 3 | 3 |
| Resulting shift | Rightshift by 3 | | |
| Certainty | 1 | | |

Table 3: Sample Not-Shift reconstruction

**Matrix-Mix**

| Samples | (101100, 110010) | | (011010, 010101) | | (110001, 001110) | |
|---|---|---|---|---|---|---|
| Possible mappings | 0, 2, 3 | 0 | 0, 3, 5 | 0 | 2, 3, 4 | 0 |
| | 0, 2, 3 | 1 | 1, 2, 4 | 1 | 2, 3, 4 | 1 |
| | 1, 4, 5 | 2 | 0, 3, 5 | 2 | 0, 1, 5 | 2 |
| | 1, 4, 5 | 3 | 1, 2, 4 | 3 | 0, 1, 5 | 3 |
| | 0, 2, 3 | 4 | 0, 3, 5 | 4 | 0, 1, 5 | 4 |
| | 1, 4, 5 | 5 | 1, 2, 4 | 5 | 2, 3, 4 | 5 |
| Resulting permutation | (0, 1, 2, 3, 4, 5) → (3, 2, 5, 1, 0, 4) | | | | | |
| Bitwise certainty | 1, 1, 1, 1, 1, 1 | | | | | |

Table 4: Sample Matrix-Mix reconstruction

## G Algorithms

**Algorithm 1** binary_search

```
Input: scene: Scene, ray: Light-ray, precision: Decimal
Output: point | None
1:  start ← ray.position
2:  direction ← ray.direction
3:  stop ← predicted exit point with no interaction
4:  actual-out ← scene.Encrypt(ray).position
5:  mid ← None
6:  if stop == actual-out then
7:      return None
8:  end if
9:
10: while start.distance(stop) > precision do
11:     mid ← start + ((stop - start)*0.5)
12:     newray ← Ray(mid, direction)
13:     newout ← scene.Encrypt(newray)
14:     if newout.distance(actual-out) == 0 then
15:         start ← mid
16:     else
17:         stop ← mid
18:     end if
19: end while
20: return mid
```

**Algorithm 2** Linear Object Reconstruction

```
    Input: samples: list[tuple(str, str)]
    Output: corners:list[tuple(str, str)]
    All samples must share the same length
 1: lineEquations ← []
 2: for each pair of samples: do
 3:     lineEq ← getLineEquation(pair)
 4:     if coversMoreThanThreePoints(lineEq, samples) then
 5:         lineEquations.append(lineEq)
 6:     end if
 7: end for
 8: for lineEq in lineEquations: do
 9:     distantPoint ← findDistantPoint(lineEq, samplePoints)
10:     distantPoints.append(distantPoint)
11: end for
12: intersections ← []
13: for each pair in lineEquations: do
14:     if calculateIntersection(pair) then:
15:         corners.append(calculateIntersection(pair))
16:     end if
17: end for
18: corners ← removeRedundantCorners(corners)
19: if |corners| < 4 then
20:     corners ← constructChain(corners, distantPoints)
21: end if
22: return corners, isopenObject
```

**Algorithm 3** Objects of higher Dimensionality Reconstruction

```
    Input: samples: list[tuple(str, str)], x0: tuple(str, ...)
    Output: coefficients: list[str, ...], residual: list[str, ...]
    All samples must share the same length
 1: for sample in samples: do
 2:     A.append(SubstituteDPIntoPolynomial(sample))
 3: end for
 4: fx ← CalcResiduals(A, x0)
 5: P, L, U ← PLU-Decomposition(A, fx)
 6: y ← forwardSubstitution(b, P, L)
 7: delta ← backwardSubstitution(y, L)
 8: coefficients ← init - delta
 9: error ← functionValue(coefficients)
10: return coefficients, error
```

**Algorithm 4** Decide Objects during Attack

```
    Input: samples: list[tuple(str, str)], box: boxObject
Output: coefficients: list[tuple(str, ...)], objectType: int
 1: samples, lightRays ← findSamplesWithBinarySearch(box)
 2: if isLinearObject(samples) then:
 3:     coefficients ← linearObjectReconstruction(samples)
 4:     objectType ← 1
 5: end if
 6: samples ← applyOffsetToSamples(box, samples)
 7: sampleSet ← findDistant(samples)
 8: solutions2D ← []
 9: solutions3D ← []
10: for sampleSubset in range (sampleSet) do
11:     solutions2D.append(solve2D(sampleSubset))
12:     solutions3D.append(solve3D(sampleSubset))
13: end for
14: minimalSolution2D ← minimalError(solutions2D)
15: minimalSolution3D ← minimalError(solutions3D)
16: if arbitraryEdgesCheck(minimalSolution3D, lightRays) then
17:     coefficients ← minimalSolution2D
18:     objectType ← 2
19: end if
20: if Not arbitraryEdgesCheck(minimalSolution3D, lightRays) then
21:     coefficients ← minimalSolution3D
22:     objectType ← 3
23: end if
24: return coefficients, objectType
```

**Algorithm 5** xor_reconstruction

```
    Input: samples: list[tuple(str, str)]
    Output: str, float
    All samples must share the same length
 1: constants ← []
 2: certainty ← 0
 3: for pair in samples do
 4:     const ← bitwse XOR (pair[0], pair[1])
 5:     constants.append(const)
 6: end for
 7: result ← bitwise majority gate over all elements in constants
 8: for const in constants do
 9:     for i dondex in range(len(result)):
10:         if const[index] == result[index] then
11:             certainty ← certainty + 1
12:         end if
13:     end for
14: end for
15: certainty ← certainty/(len(result) · len(constants))
16: return result, certainty
```

**Algorithm 6** notshift_reconstruction

```
    Input: samples: list[tuple(str, str)]
    Output: str, float
    All samples must share the same length
 1: constants ← []
 2: certainty ← 0
 3: for pair in samples do
 4:     neg ← bitwise negated pair[0]
 5:     for amount in range(len(pair[0])) do
 6:         shifted ← neg cyclicly shifted by 1 to the right
 7:         if shifted == pair[1] then
 8:             constants.append(amount)
 9:         end if
10:     end for
11: end for
12: result ← value occurring most often in constants
13: for pair in samples do
14:     neg = bitwise negated pair[0]
15:     shifted = neg shifted by result places to the right
16:     if shifted == pair[1] then
17:         certainty ← certainty + 1
18:     end if
19: end for
20: certainty ← certainty/len(samples))
21: return result, certainty
```

**Algorithm 7** permutation_reconstruction

```
    Input: samples: list[tuple(str, str)]
    Output: list[int] | None, float
    All samples must share the same length
 1: index_lists = []
 2: for orig_ind in range(len(samples[0][0])) do
 3:     cur_list ← []
 4:     for pair in samples do
 5:         ind_set ← set()
 6:         for end_ind in range(len(pair[1])) do
 7:             if pair[0][orig_ind] == pair[1][end_ind] then
 8:                 ind_set.add(end_ind)
 9:             end if
10:         end for
11:         cur_list.append(ind_set)
12:     end for
13:     index_lists.append(cur_list)
14: end for
15:
16: result ← []
17: certainty ← 0
18: for ind in range(len(index_lists)) do
19:     intersected ← intersection of all sets per index
20:     if len(intersected) == 0 then
21:         return None, 0
22:     end if
23:     certainty ← certainty + (1/len(intersected))
24:     result.append(intersected.pop)
25: end for
26: certainty ← certainty/ len(index_lists) return result, certainty
```

**Algorithm 8** clustering_given_points

```
Input: box_dict: dict(list(point)), width: float, height: float
Output: dict[box_id](Box)

out ← empty dict
for box_id in box_dict.keys() do
    points ← box_dict[box_id]
    x_min, x_min_c ← minimal x-coordinate, nr. of occurrences of that value
    x_max, x_max_c ← maximal x-coordinate, nr. of occurrences of that value
    y_min, y_min_c ← minimal y-coordinate, nr. of occurrences of that value
    y_max, y_max_c ← maximal y-coordinate, nr. of occurrences of that value
    pos_x, pos_y = 0, 0
    x, y = False, False
    if x_min_c > 1 then
        pos_x ← x_min
        x ← True
    else if x_max_c > 1 then
        pos_x ← x_max - width
        x ← True
    end if

    if x_min_c > 1 then
        pos_y ← y_min
        y ← True
    else if x_max_c > 1 then
        pos_y ← y_max - height
        y ← True
    end if

    if x == True and y == True then
        out.add(Box((pos_x, pos_y), (width, height)))
    else
        gather more samples, retry reconstruction
    end if
end for
return out
```