Introducing Gate Based Ray Tracing Cryptography

Sneha Mohanty, Eric Peairs and Christian Schindelhauer

June 28, 2023

INTRODUCING GATE BASED RAY TRACING CRYPTOGRAPHY

TECHNICAL REPORT

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

Eric Peairs
Computer Networks and Telematics
University of Freiburg
Freiburg, Germany
eric.peairs@uni-freiburg.de

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

June 23, 2023

ABSTRACT

We present a novel Gate based symmetric cryptographic system in a 2D environment that incorporates ray tracing on standard conics, such as; parabola, hyperbola, ellipse etc. as well as on generic first, second and third degree polynomials in order to encrypt as well as decrypt information using a light ray. In our scheme we also make use of discrete boolean logic gates in a pseudo-random manner on the light ray at different parts of the 2D environment to make the cryptographic scheme even more complex and secure.

Keywords Objects · Ray tracing · Cryptography · Gates

1 Introduction

We introduce a Gate based symmetric key Cryptographic scheme in a 2D environment (in cartesian coordinates) consisting of multiple objects in the form of standard conics such as parabola, ellipses, hyperbola etc. as well as generic first, second and third degree polynomials. At each step of the encryption process, the scheme is made secure due to the interaction of the light ray with linear and non-linear equations in the form of the aforementioned objects, involving reflection or refraction. The security is further enhanced due to the projection of the light ray after each of these interactions into a new Non-optical gate box position in the 2D environment depending on the boolean Gate based operation applied to the light ray prior to each of it's projection.

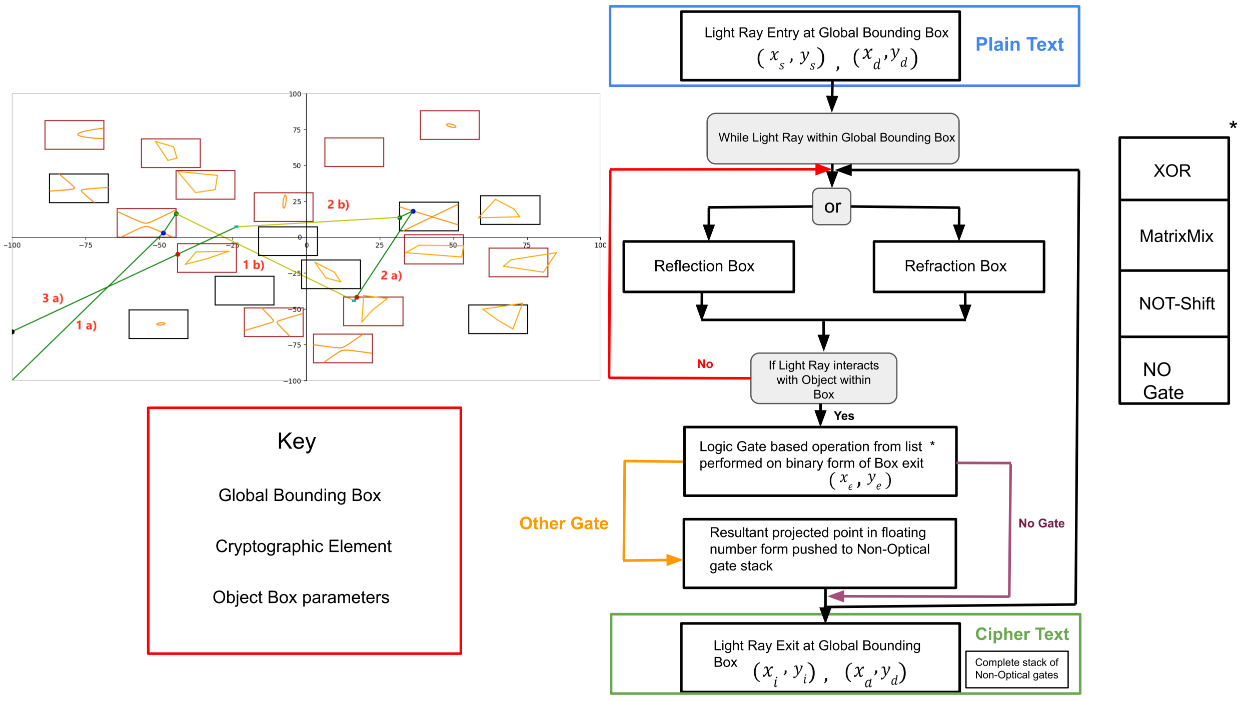
Motivation In the past years, studies have been conducted in the area of Ray Tracing as well as Cryptography. We introduce a novel technique here to combine both of these areas together in order to present the idea that ray tracing involving discrete and non-linear transformations of the position and direction of the light ray in a 2D environment can be used successfully as a symmetric key cryptographic scheme. Since this approach is newly introduced by us, combining two vast areas of research in a very specific manner, we can arguably consider our cryptographic scheme to be highly secure.

2 Related Work

Our work is inspired mainly from Reif et al. [1994] wherein the ray tracing problem takes a initial light ray at a certain position and depending on the configurations of various objects in the 2D setup (optical system), it is determined whether or not the final light ray exits at a fixed point, p. 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 is undecidable. Han et al. [1999] worked on Optical image Encryption based on XOR operations. Blansett et al. [2003] discusses 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.

3 Overview

Figure 1 showcases a sample 2D setup with all our visual components as well as the part of the Encryption performed due to subsequent reflection or refraction at the individual surfaces of these visual components. The part of our Encryption scheme involving discrete boolean logic gates has also been shown here.



Light Ray Entry at Global Bounding Box
(x_s, y_s), (x_d, y_d)
Plain Text
While Light Ray within Global Bounding Box
or
Reflection Box
Refraction Box
No
If Light Ray interacts with Object within Box
Yes
Logic Gate based operation from list * performed on binary form of Box exit (x_e, y_e)
Other Gate
No Gate
Resultant projected point in floating number form pushed to Non-Optical gate stack
Cipher Text
Light Ray Exit at Global Bounding Box (x_i, y_i), (x_a, y_d)
Complete stack of Non-Optical gates
*
XOR
MatrixMix
NOT-Shift
NO Gate
Key
Global Bounding Box
Cryptographic Element
Object Box parameters
1 a)
1 b)
2 a)
2 b)
3 a)

Figure 1: Encryption steps of the Gate based Symmetric key Cryptographic scheme

4 Visual Elements

Our approach takes a ray of light starting at the boundary of the environmental setup, (x_s, y_s) and let's it hit various objects in the given 2D coordinate system. The objects that are placed in various parts of this given environment are intersected, while keeping the incident and reflection angles equal in the respective cases if reflection is chosen. In the case of refraction, the light ray bends at a certain angle at the boundary of these objects depending on the Snell's law and the refractive indices of the respective media. We are aware that the laws of reflection and refraction are different for curved surfaces in comparison to plane surfaces but to maintain uniformity in the case of generic and standard reflective as well as refractive surfaces, we use the techniques mentioned in the paragraph above, for reflection and refraction respectively.
Throughout our work, we have made use of 'objects' in the form of standard and generic conics in 2D of degree one, two and three respectively. A sample case of visual elements in our 2D environment has been shown in Figure 1. We have covered objects that could also be translated and rotated w.r.t the cartesian coordinate system. Table 1 illustrates these objects in a compact form.

Local Object Box This consists at max of one standard or generic conic or polygon. This means that it can consist of either no object or a single object. For our Gate based cryptography approach, we have introduced two types of Local Object Boxes, i.e; a 'Reflection Box' which signifies that reflection occurs at the surface of the object contained within it if the light ray intersects with it as well as a 'Refraction Box' wherein refraction occurs if the light ray intersects with the object contained within it. The Local Object Box also serves as a boundary restricting the objects described in Table 1. In the absence of this boundary, the object would be allowed to stretch infinitely within the bounds of the Global Object Box, thereby potentially causing issues for the encryption-decryption process.

Global Object Box This box, also referred to here as Global Bounding Box in our scheme consists of the complete 2D environment in (x, y) cartesian coordinates that could contain multiple Local Object Boxes within it.

Object The Objects are as described in the subsections below. These are either customizable in that the user renders the objects at specific locations of the 2D environment or are dynamically generated as part of the encryption key in the Gate based approach developed by Tobias Grugel as part of his Master Thesis (Grugel [2023]).

4.0.1 Standard and Generic objects in 2D

The table below illustrates standard conics (Parabola, Ellipse, Hyperbola) as well as a linear equation in 2D. It also illustrates the generic second as well as third degree polynomials in the 2D cartesian system.

Object Type	Equation
Linear	$d_y(x - x_0) - d_x(y - y_0) = 0$
Parabola	$ax'^2 - y' = 0$
Ellipse	$x'^2/a^2 + y'^2/b^2 = 1$ $b^2x'^2 + a^2y'^2 - a^2b^2 = 0$
Hyperbola	$x'^2/a^2 - y'^2/b^2 = 1$ $b^2x'^2 - a^2y'^2 - a^2b^2 = 0$
Generic Second Degree Polynomial	$ax'^2 + by'^2 + cx'y' + dx' + ey' + f = 0$
Generic Third Degree Polynomial	$ax'^3 + by'^3 + cx'^2y' + dx'y'^2 + ex'^2 + fy'^2 + gx'y' + hx' + jy' + k = 0$

Table 1: **Objects in 2D**

These objects are put through rotation and translation matrices in 2D using the following rotation and translation matrices respectively :

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} cos\theta & sin\theta \\ -sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} x - h \\ y - k \end{bmatrix} = \begin{bmatrix} (x - h)cos\theta + (y - k)sin\theta \\ -(x - h)sin\theta + (y - k)cos\theta \end{bmatrix}$$

The rotation matrix rotates the objects in the counter-clockwise direction relative to the space.

4.1 Light Ray

The other major visual element in our setup is the light ray itself. This light ray is a vector, with the form described by Eq. (1), 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) 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 :

$$\begin{pmatrix} x_s \\ y_s \end{pmatrix} + \lambda \begin{pmatrix} x_d \\ y_d \end{pmatrix} \tag{1}$$

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 λ if the intersection of the light ray and an object is *behind* the source of the light ray. The positive λ and negative λ are derived from solving linear, quadratic and cubic equations in our case, as illustrated in the following sections.

4.2 Intersection Calculation

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

4.2.1 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 λ.

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

4.2.2 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.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} cos\theta & sin\theta \\ -sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} (x_s + \lambda x_d) - h \\ (y_s + \lambda y_d) - k \end{bmatrix} = \begin{bmatrix} cos\theta & sin\theta \\ -sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} x_s - h \\ y_s - k \end{bmatrix} + \lambda \begin{bmatrix} cos\theta & sin\theta \\ -sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} x_d \\ y_d \end{bmatrix} \tag{3}$$

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 λ is then found after substituting the new x' and y' into the object's equation.

This results in a quadratic equation in λ.

$$a\lambda^2 + b\lambda + c = 0 \tag{4}$$

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.

4.2.3 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 λ.

$$a\lambda^3 + b\lambda^2 + c\lambda + d = 0 \tag{5}$$

To solve the cubic equation we use *Cardano's formula* whose usage has been illustrated below.

$$\begin{aligned} \mathcal{Q} &\equiv \frac{3ac - b^2}{9a^2} \\ \mathcal{R} &\equiv \frac{9abc - 27a^2d - 2b^3}{54a^3} \\ \mathcal{D} &\equiv \mathcal{Q}^3 + \mathcal{R}^2 \\ \mathcal{S} &\equiv \sqrt[3]{\mathcal{R} + \sqrt{\mathcal{D}}} \\ \mathcal{T} &\equiv \sqrt[3]{\mathcal{R} - \sqrt{\mathcal{D}}} \end{aligned} \tag{6}$$

The roots of the cubic equation are given by

$$\begin{aligned} \lambda_1 &= -\frac{b}{3a} + \mathcal{S} + \mathcal{T} \\ \lambda_2 &= -\frac{b}{3a} - \frac{1}{2}(\mathcal{S} + \mathcal{T}) + \frac{i\sqrt{3}}{2}(\mathcal{S} - \mathcal{T}) \\ \lambda_3 &= -\frac{b}{3a} - \frac{1}{2}(\mathcal{S} + \mathcal{T}) - \frac{i\sqrt{3}}{2}(\mathcal{S} - \mathcal{T}) \end{aligned} \tag{7}$$

discriminant	roots
$\mathcal{D} < 0$	3 real, all unequal
$\mathcal{D} = 0$	3 real, at least 2 are equal
$\mathcal{D} > 0$	1 real, 2 complex

Table 2: Roots of cubic formula

Similar to the second degree object, only the real roots of the cubic equation are of interest to us. The polynomial discriminant can give us an indication how many real roots we could expect.

In the case when $\mathcal{D}$ is negative, the programming language can no longer find values for $\mathcal{S}$ or $\mathcal{T}$ and therefore cannot calculate the roots. In this case, we use De Moivre's Theorem in order to rewrite the equation into something that can be directly calculated. Solving for $\mathcal{S}$ as an example, $\mathcal{S} = \sqrt[3]{\mathcal{R} + i\sqrt{|\mathcal{D}|}} = G^{1/3} * (\frac{\mathcal{R}}{G} + i\frac{\sqrt{|\mathcal{D}|}}{G})^{1/3} = G^{1/3} * (cos\theta + isin\theta)^{1/3}$. At this point, we apply De Moivre's Theorem to get $G^{1/3} * (cos(\theta/3) + isin(\theta/3))$. This can then be simplified by removing θ and reducing the trigonometric functions. $G^{1/3} * e^{i\theta/3} = G^{1/3} * (e^{i*atan(\sqrt{|\mathcal{D}|}/\mathcal{R})/3})$. Here, we define $G = (\sqrt{\mathcal{R}^2 + (\sqrt{|\mathcal{D}|})^2})$ and $\theta = atan(\frac{\sqrt{|\mathcal{D}|}}{\mathcal{R}})$ for simplicity.

4.3 Tangent and Normal

4.3.1 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.

4.3.2 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, ∇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 $-\vec{n}$.

4.4 Reflection at the Point of Interest

Reflection at the point of interest, i.e; the point of intersection between the object and the light ray in our case has been calculated using the mirroring technique elaborated below.

Mirror Technique The Mirror technique evaluates the mirrored point of the incident light ray taking the tangent as the mirroring element. The mirrored point is then traced through the point of intersection with the object, (x_p, y_p) in order to find the equation of the reflected light ray. This technique has been explained in detail in the Appendix A. An example illustration of reflection has been shown in Figure 2. In our scheme, the brown box could be used as an additional boundary, to limit the object within it. It could also be used interchangeably with the Local Bounding Box, serving the same purpose, as described in Section 4. The Local Bounding box specifically in this setup, in Figure 3 as well as in Figure 6 shows us where the object is centered in it's local environment.

Using $tan\theta$ This technique mainly uses the concept of $\tan\theta$ calculation to make sure that the angle of incidence is equal to the angle of reflection at the intersection point with the object, (x_p, y_p). This information is then used to calculate the equation of the reflected light ray.

Vector Reflection The reflection can also be described in terms of a vector, $\vec{r} = \vec{i} - 2(\vec{i} \cdot \hat{n})\hat{n}$. Here, the input vector $\vec{i}$ is projected onto the normalized normal vector, $\hat{n}$. Subtracting this value in the $\hat{n}$ direction creates a right triangle. Subtracting this again creates an isosceles triangle with sides $\vec{i}$, $2(\vec{i} \cdot \hat{n})\hat{n}$, and $\vec{i} - 2(\vec{i} \cdot \hat{n})\hat{n}$. This final side is parallel to the output vector. This process is explained more elaborately in the source by de Greve [2004]. It is also elaborated in the Appendix C.

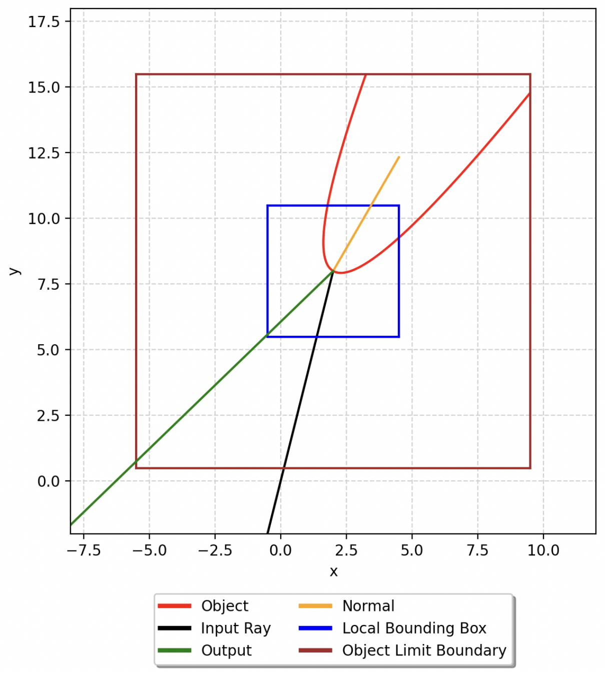


Figure 2: Example of reflection of ray at the surface of a parabola

4.5 Refraction at the Point of Interest

Refraction of light through two mediums is calculated using Snell's Law.

$$n_1 sin\theta_1 = n_2 sin\theta_2 \tag{8}$$

However, this can be re-written to exclude any angles and only use vectors to calculate an output vector de Greve [2004]. Here, the two mediums are combined into a single term, $\mu = n_1/n_2$.

$$\vec{r} = \mu(\hat{i} - (\hat{n}\cdot\hat{i})\hat{n}) + \hat{n}\sqrt{1 - \mu^2(1 - (\hat{n}\cdot\hat{i})^2)} \tag{9}$$

. The Vector based technique for refraction has been covered in detail in the Appendix D. An illustration of refraction at the surface of an object has been shown here in Figure 3.

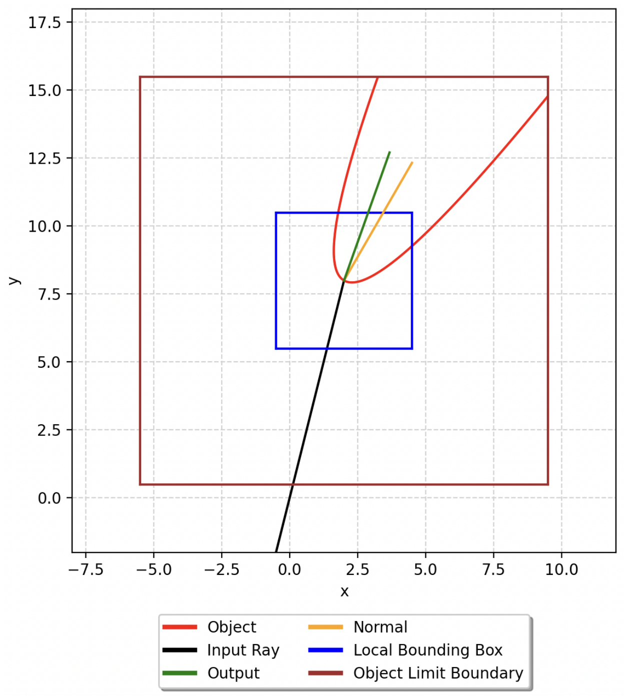


Figure 3: Example of refraction of ray through a parabola

5 Gate based Symmetric Cryptographic Scheme

5.1 Encryption

The Encryption process as described using the Flow Chart in Figure 1 begins at the initial position of the light ray, (x_s, y_s), with initial direction, (x_d, y_d), located at the boundary of the Global Bounding Box. These parameters also form the Plaintext of our cryptographic scheme. On entering a Local Object Box (either Reflection or Refraction Box) and having performed intersection with an object and thereafter successful reflection or refraction, the exit point of the light ray, i.e; the point of intersection of the reflected or refracted light ray with the Local Object Box, (x_e, y_e) is obtained. This floating point coordinate is transformed into it's binary representation. The binary (x, y) coordinate obtained at this step is then passed through a boolean logic gate of type XOR, MatrixMix, rotated NOT-Shift or NO Operation in a pseudo-randomized manner of selecting one of these gates. The resulting point is then transformed from it's binary representation to it's floating point representation, which gives us the projected point, $P' = (x', y')$. We create a Non-Optical gate Box at this point of projection and retain the same direction as the reflected/refracted light ray in the previous Local Object Box, where the light ray originally exited, at (x_e, y_e). This Non-Optical gate Box is pushed to a stack. No operation is performed on the exit point of the Local Object Box, (x_e, y_e) when none of the Non-Optical gate coordinates using the above logic gate operations turn out to be valid. The conditions for the validity of the Non-Optical gate point is described in the subsequent paragraphs . The new light ray then either intersects the next Local Object Box containing the next object in the 2D environment (in the case where one of the aforementioned boolean logic gates were used) or continues onward without any changes after reflection or refraction in the Local Object Box (in the case where No operation was performed on the light ray). The whole process is now repeated until the light ray finally intersects the Global Bounding Box and thereafter marks an exit point, (x_i, y_i) and exit direction, (x_d, y_d). These form the Ciphertext of our encryption scheme along with the complete stack of Non-Optical gate boxes. There could also potentially be cases where the light ray doesn't reach the Global Bounding Box even after multiple successive intersections with objects as well as projections to Non-Optical Box positions. For such cases, the number of interactions of the light ray with the objects can be custom chosen, thereby leading to an 'early' final exit point (x_i, y_i) and final direction, (x_d, y_d) before reaching the Global Bounding Box. A sample Encryption over our 2D setup is shown in Figure 1 where 1 a) denotes the incoming light ray, into the Global Bounding Box, 1 b) shows the projection of this light ray to the first Non-Optical gate position, after refracting in the first Local Bounding Box. 2 a) shows the new projected light ray entering the next Local Object Box, 2 b) denotes the projection of this light ray to a new

Non-Optical gate Box position after reflection and 3 a) shows the final light ray in this 2D scene, which exits at the Global Bounding Box.

Non-Optical Gate Box This is a box within the Global Bounding Box. It does not contain any object nor is it considered during the encryption process (except that 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 5.2, 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).

Validity of Non-Optical Gate Box 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.

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.

5.2 List of Discrete Boolean Gates

Based on pseudo-random selection, one of XOR, MatrixMix or NOT-Shift operation is chosen to be applied on the point of exit, (x_e, y_e) of the light ray at the Local Object Box, after transforming these coordinates from their floating point form to their binary form, provided the point of projection, $P' = (x', y')$ (position of the Non-optical gate box) turns out to be valid. The algorithms used to perform floating point decimal to binary conversion and vice-versa have been shown in Appendix E. All of the gate based operations that we use are invertible. This property has been used in designing our symmetric key cryptographic scheme.

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

The XOR-Gate takes two bit-numbers as input: Let a, b be 8-bit numbers, we compute the XOR $a \oplus b$ in a bit-wise manner. That means we XOR the same position of the two numbers to get the resulting XORed value.

$$a[i] \oplus b[i] = c[i] \text{ where } 0 \leq i \leq 7 \tag{10}$$

The resulting value c is an 8-bit number 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 Let a, x be a bit-number

$$(a \oplus x) \oplus x = a \oplus (x \oplus x) = a \oplus 0 = a \tag{11}$$

MatrixMix The MatrixMix-Gate is the operation where we could potentially mix every position of a bit-number a. The input of the MatrixMix-Gate is the matrix representation of the bit-number a and a mixing matrix M.

The mixing matrix is a pre-computed permutation of all matrix positions for a given matrix size. The possible matrix sizes used in our scheme have been shown in the Appendix F.

Let M be a 2×3 matrix, the positions of a 2×3 are the following

$$\begin{pmatrix} (0,0) & (0,1) & (0,2) \\ (1,0) & (1,1) & (1,2) \end{pmatrix} \tag{12}$$

To compute the mixing matrix, we take all positions of a matrix as a list, compute a permutation of this list, and transform the permutation back to a matrix. To invert the process, we follow the reverse steps until we obtain back the original bit input.

$$\begin{aligned}
&\text{Original:}(0,0),(0,1),(0,2),(1,0),(1,1),(1,2) \\
&\text{MatrixMix:}(1,2),(1,0),(1,1),(0,0),(0,1),(0,2) \\
&\text{Dict(MatrixMix):}(0,0):(1,2),(0,1):(1,0),(0,2):(1,1),(1,0):(0,0),(1,1):(0,1),(1,2):(0,2) \\
&\begin{pmatrix} (1,2) & (1,0) & (1,1) \\ (0,0) & (0,1) & (0,2) \end{pmatrix}
\end{aligned} \tag{13}$$

$$\begin{aligned}
&\text{Dict(InvMatrixMix):}(1,2):(0,0),(1,0):(0,1),(1,1):(0,2),(0,0):(1,0),(0,1):(1,1),(0,2):(1,2) \\
&\text{SortedDict(InvMatrixMix):}(0,0):(1,0),(0,1):(1,1),(0,2):(1,2),(1,0):(0,1),(1,1):(0,2),(1,2):(0,0) \\
&\text{InvMatrixMix:}(1,0),(1,1),(1,2),(0,1),(0,2),(0,0) \\
&\begin{pmatrix} (1,0) & (1,1) & (1,2) \\ (0,1) & (0,2) & (0,0) \end{pmatrix}
\end{aligned} \tag{14}$$

There are some cases where the bit-string is of prime number length, p. In such cases, we compute the mixing matrix in a slightly modified manner. We used the prime-factorization to create the size of the matrices, as seen from the Table in Appendix F. This leads to an issue when the length p is prime. If we used the prime-factorization for these cases, we end up having a matrix of size $(1, p)$ or $(p, 1)$.

To resolve this issue, we break the p length bit-string into the first $p - 1$ bits and then store the last bit of the sequence separately. We can now proceed as the normal case with the $p - 1$ bits by taking it's prime factorization and having obtained the final MatrixMix string, we add back the last bit into this new sequence.

NOT-Gate followed by Left-/Right-Rotational-Shift The NOT-Gate takes one bit-number as input: Let a be an 8-bit number, we compute NOTa in a bit-wise manner.

$$\neg a[i] = c[i] \text{ where } 0 \leq i \leq 7 \tag{15}$$

Also, in the reverse direction,

$$\neg\neg a[i] = \neg c[i] = a[i] \text{ where } 0 \leq i \leq 7 \tag{16}$$

The resulting value c is an 8-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 8-bit number, let k be a integer. We take $k \mod 8$, let the direction be left:

$$c[i] = \neg a[i + k \mod 8] \text{ where } 0 \leq i \leq 7 \tag{17}$$

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.

5.3 Key

The Key in our cryptographic scheme consists overall of three parts and is created dynamically from the terminal during runtime. A sample key has been shown in Figure 4. 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.

```
[[-100.0, -100.0, 200.0, 200.0],  \\ global bounding box parameters

[5676, 200.0], \\ parameters for the cryptographic element

[[2,              \\ object and object box parameters
"refraction",
["-18.64450907414142477591667557135224342346191406 25", "44.862656600994142763738636858761310577392578125",
"30.352067564449839665030594915151596069335937 5", "15.478447821087442548559920396655797958374023437 5",
"23.025682164289221987019118387252092361450195312 5", "15.352423956762279999566089827567338943481445312 5",
"-31.916450575792282506881747394800186157226562 5", "-21.217317730690922417124966159462928771972656 25",
"0.130336273842283301149791441275738179683685302734375", "1.00029999999999996695976278715534135699272155761 71875",
"1.520000000000000177635683940025046467781066894531 25", "1111001111101100001110000010100100101110011010100010000100110111",
"1111101001100101000100100100100110011101111011010111101100011011",

 {"(2, 2)": {"(0, 0)": "(0, 1)", "(0, 1)": "(1, 0)", "(1, 0)": "(0, 0)", "(1, 1)": "(1, 1)"},
  "(2, 3)": {"(0, 0)": "(0, 0)", "(0, 1)": "(0, 2)", "(0, 2)": "(0, 1)", "(1, 0)": "(1, 1)", "(1, 1)": "(1, 0)", "(1, 2)": "(1, 2)"},
  "(2, 4)": {"(0, 0)": "(0, 2)", "(0, 1)": "(0, 0)", "(0, 2)": "(1, 3)", "(0, 3)": "(1, 0)", "(1, 0)": "(1, 1)", "(1, 1)": "(0, 3)", "(1,
2)": "(0, 1)", "(1, 3)": "(1, 2)"},

.........,
\\ rest of the dictionary components removed due to lack of space for visibility
}, "d5420a29-c1b6-46f3-9fa5-7df50162db6e", 1, 0]]]]
```

Figure 4: Components of a sample Key

The parameters for the Global Bounding Box include it's 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), two 64 bit-strings for the XOR gate, each a string of random bits to use as the respective mask for the part of the XOR number before the decimal place and the other for the part of the XOR number after the decimal place, followed by dictionaries for the MatrixMix-Gate as well as Inverse MatrixMix-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.

5.4 Decryption

The Decryption process for the same setup as in Figure 1 is shown in Figure 5. Decryption begins at the final position of the light ray, (x_i, y_i), with final direction, (x_d, y_d), located at the boundary of the Global Bounding Box. We also return the full stack of Non-Optical gate Boxes. These form the Ciphertext of our symmetric key Cryptographic scheme. We begin the decryption process by popping the last Non-Optical gate Box element from the stack, thereafter followed by converting the Non-Optical gate box position coordinates from it's floating point representation to binary. The binary form of these coordinates are then transformed using the Inverse operation of the corresponding boolean logic Gate, that was applied during the Encryption process. This leads back to the previous exit point, (x_e, y_e) of the reflected or refracted light ray at the Local Object Box. Then using the laws of reflection or refraction, depending on the type of the Local Object Box, the light ray is retraced backwards. This whole process continues until all the Non-Optical gate Boxes are popped from the stack and the corresponding inverse Gate based operations have been applied, leading to the initial position of the light ray, (x_s, y_s) and initial direction, (x_d, y_d) where the light ray first began at the Global Bounding Box. The numbers 1, 2 and 3 respectively in the Figure 5 below show the subsequent steps in our symmetric key cryptographic scheme for a sample 2D scene.

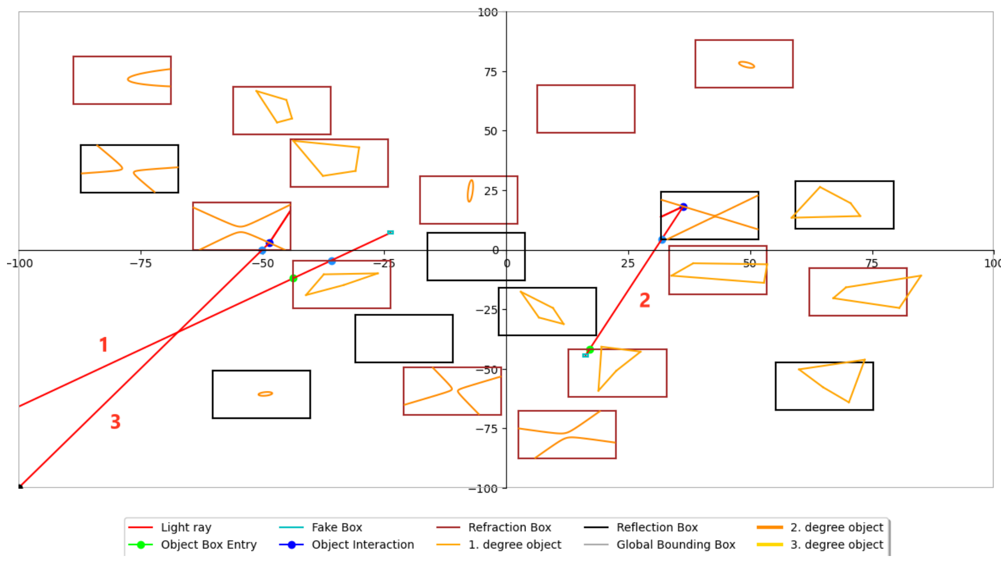


Figure 5: Decryption steps of the Gate based Symmetric key Cryptographic scheme

5.5 Precision of values

For our simulation, we have split the floating point representation of the point where the light ray exits a Local Bounding Box (x_e, y_e) after having interacted with the object within it in the form of reflection or refraction, into two parts, the part before the decimal point (integer part) and the part after the decimal point (decimal part). Both the x_e and y_e are split in the aforementioned manner. Thereafter we set the number of default positions after the decimal point to be 15. So, the maximum number that can be fit into it is 999999999999999 (which can be represented using around 50 bits). For the part before the decimal point, the total number of positions is set as default to the $\log_2(widthofGlobalBoundingBox)$.

In the next step, while converting the floating point number described above to it's binary representation, we set the first bit of the part before the decimal point to be 0 or 1 depending on whether the number in it's floating point representation is positive or negative respectively.

The key described in Section 5.3 has bit-strings of length 64 each for XOR masking of the part before the decimal point as well as the part after the decimal point so that enough bits are available to mask the binary form of (x_e, y_e) at the Local Object box exit, incase XOR gate is selected using the pseudo-random selection of gates described in Section 5.2. The unused bits of the XOR masks are just stored as noise and don't partake in the rest of our cryptographic scheme.

In practice, it is possible to have any number of positions on the left as well as the right of the decimal place of the floating point numbers, (x_e, y_e) but for a large number of interactions of the light ray in the 2D environment, we can expect a slight difference between the actual initial start point of the light ray, (x_s, y_s) and the computed position of the light ray obtained after decryption. This divergence factor could also be dependent on the Programming language used for simulation.

6 Attacks and Mitigation

This section elaborates the possible attacks to our cryptographic system and their respective mitigation techniques.

6.1 Pattern Detection

The attacker could launch a brute-force attack on the 2D environment to learn the objects. While it might be very difficult to learn 'open' objects, since not only does the light ray need to retrace it's way backwards during decryption through Non-optical gate boxes but also has to solve potentially several linear and non-linear equations during the reflection or refraction processes to reach the initial start point of the light ray, (x_s, y_s). However, over time, the structure of 'closed' objects, such as; ellipses or polygons have some possibility of getting learnt since no Non-Optical gate box could be created in such parts of the 2D environment.

The other possibility of Pattern Detection is when only one of the objects is intersected by the initial light ray, involving reflection or refraction before reaching the Global Bounding Box as an exit point. This makes the 2D setup susceptible to both, Plaintext as well as Ciphertext attacks, thereby leading to the discovery of the object position as well as shape. Therefore we would ideally need atleast two interactions of the light ray in the 2D environment to make our cryptographic scheme secure enough.

6.2 Plaintext Attack

As described in Section 5.1, the Plaintext in our scheme consists of the initial origin of the light ray, (x_s, y_s) and it's initial direction, (x_d, y_d) at the Global Bounding Box. Even if the attacker has this information, they would not be able to recreate the 2D environment with the Local Object Boxes, the object positions, the objects themselves, the unique identifiers linking the correct Non-Optical gate boxes to their original Local Object boxes as well as the light ray path through them since all of the above information is part of the key itself, as described in Section 5.3. Therefore as far as the possibility of a Plaintext attack goes, our cryptographic scheme stands arguably robust against it.

6.3 Ciphertext Attack

As described in Section 5.4, the Ciphertext in our scheme consists of the final exit point of the light ray, (x_i, y_i) and it's final direction, (x_d, y_d) at the Global Bounding Box. It also includes the full stack of Non-Optical gate boxes produced during the encryption process.

Intercepting this information may let the attacker know about the positions of the Non-Optical gate Boxes, but retracing the way back through the objects is not possible without the knowledge of the Gate-based operation linked to each Non-Optical gate Box. Also each Non-Optical gate Box comes with it's unique identifier, that links it back to it's original Local Object Box. These critical information are only part of the key itself. The attacker might then try to brute-force their way back to the original Local Object Boxes from the respective Non-Optical gate Boxes but there is every possibility that they would not reach the Global Bounding Box at the correct initial position of the light ray, (x_s, y_s). At this point the stack of Non-optical gate boxes would already be empty and the Plaintext is not known to the attacker, so it's arguably difficult for the attacker to verify whether the position they reached at the Global Bounding Box is indeed the actual initial position of the light ray.

7 Applications

One possible application of our visual toolbox even without the discrete logic Gate-based elements could be to use the objects and Local Object Boxes in order to enhance existing cryptographic schemes such as the Hybrid ECC model, the Flow diagram of which has been shown in the Appendix G. The final exit point of the light ray, (x_i, y_i) at the Global Bounding Box or in the case of customization, at the end of the specified number of object interactions could be fed into the Hybrid ECC model as part of the $plaintext$. The other parameters that could be part of the $plaintext$ are the final direction of the light ray, (x_d, y_d) at the Global Bounding Box, the magnitude (Euclidean distance) of the initial ray between the first intersection with an object and the initial point of the light ray, and the number of interactions that have occurred before the final point was reached by the light ray. Once a shared key has been generated using the Elliptic curve cryptography (ECC) scheme, Advanced Encryption Standard (AES) could be used to encrypt each of the aforementioned $plaintext$ parameters. The AES encryption creates three outputs: $ciphertext$, $authenticationTag$, and $nonce$.

The $ciphertext$ is the encrypted input message. In order to check for tampering, $authenticationTag$ is generated using the $ciphertext$ as an input for an authentication function. The value is sent to the receiver, who runs the same authentication function using the $ciphertext$ received as an input. When the value of the $authenticationTag$ received does not match the value of the $authenticationTag$ generated, it is a sign the $ciphertext$ has been tampered during transmission. When an AES system is set up, a random value is required during the setup. This value is $nonce$ and must be sent as an output in order for the receiver to setup their decrypting AES system with the same random number.

The receiver could then use these as well as $publicKeySender$ and the $privateKeyReceiver$ generated during the ECC encryption step to decrypt the message and ensure it has not been tampered during transmission. The $ciphertext$ in this case is a binary number generated for each of the $plaintext$ parameters described above. These binary numbers could be changed into their floating point representations and then bounded within the Global Bounding Box limits using modulo operations before then multiplying by a randomly selected positive or negative 1 in order to 'fool' the attacker into believing these to be indicative of the true point of exit of the light ray.

Figure 6 shows a sample 2D setup with the initial light ray in green, followed by interactions of the light ray with multiple objects and the final light ray in yellow. We have customized the number of interactions here of the light ray with the objects, therefore the final light ray doesn't necessarily exit at the Global Bounding Box.

In the case of a Plaintext attack on the 2D setup, with the plaintext involving the initial light ray position and direction, the attacker would have to potentially solve multiple interactions with linear and non-linear objects in the form of reflections or refractions before arriving at the correct final direction and exit point of the light ray (therefore, the ciphertext). But these ciphertext parameters after the ray-tracing through the visual elements in the 2D setup could be further fed into the Hybrid ECC model as plaintext parameters, in the manner described in the paragraphs above. The final exit point of the light ray, following the Hybrid ECC scheme would be the 'purple dot' such as the one shown in Figure 6. This would arguably throw off an attacker into assuming that this is the actual exit point of the light ray.

Also, another method to 'fool' the attacker attempting a Plaintext attack on the 2D setup could be to 'mute' some objects, such as; the parabola shown in Figure 6 doesn't actually interact with the initial light ray. This could 'fool' the attacker into pursuing a completely different path in the 2D setup and thereafter exit at a wrong point.

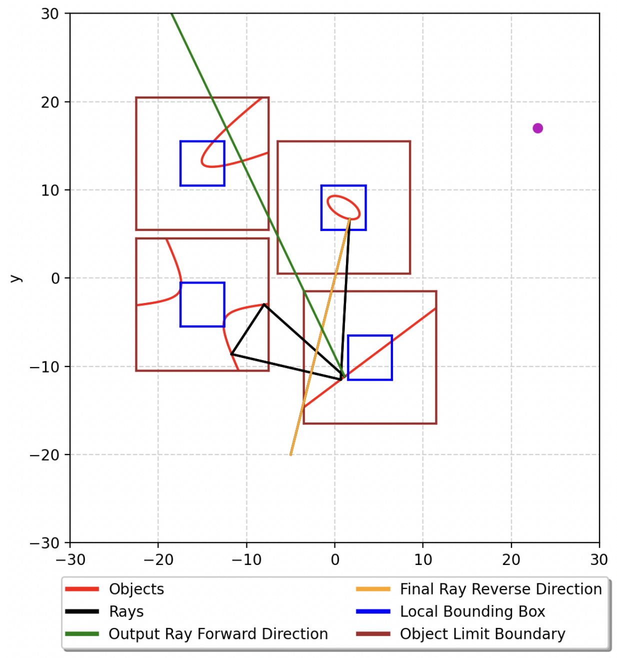


Figure 6: An example 2D setup including the false exit point

8 Conclusion and Future Work

We have introduced a novel Gate-based symmetric key cryptographic approach to encrypt and decrypt information on a light ray in a 2D ray tracing environment. We have used reversible logic Gate based as well as reversible reflection and reflection operations, thereby making our cryptographic scheme symmetric. The encryption process is not only made secure because potentially multiple non-linear as well as linear equations have to be solved in order to trace the path of the light ray through the objects, but is further enhanced by incorporating pseudo-random selection of discrete logic gate based operations to be applied at the exit point, (x_e, y_e) of every Local Object Box. A further layer of security is ensured by producing the 2D setup dynamically through the terminal at every run. Added to that is the security that comes from the fact that the XOR-masks, the MatrixMix permutation dictionaries as well as the shifting direction and the shift constant for the NOT-Shift gate are all part of our key, and are hence hidden information. We also discuss about the possible errors in precision values in our scheme due to divergence in floating point numbers in Section 5.5. In Section 6 we elaborate different types of attacks and their possible mitigation strategies. In Section 7, we finally present an application of our 2D setup in combination with the Hybrid ECC model to create a complex cryptographic scheme.

As a possible Future Work, we would like to incorporate our Gate based cryptographic system into a 3D environment, consisting of (x, y, z) cartesian coordinates. We would also like to design and launch specific attacks on our existing cryptographic scheme in 2D and study it's robustness against these. Apart from this we would be exploring the possibility of adding both reflection as well as refraction simultaneously on the surfaces of our 2D objects using Fresnel equations as described in Skaar [2019].

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.

Bram de Greve. Reflections and refractions in ray tracing. 2004.

Ecc encryption/decryption. https://cryptobook.nakov.com/asymmetric-key-ciphers/ecc-encryption-decryption. Accessed: 2023-06-23.

Johannes Skaar. Fresnel's equations in statics and quasistatics. *European Journal of Physics*, 40(4):045201, jun 2019. doi:10.1088/1361-6404/ab166b. URL https://dx.doi.org/10.1088/1361-6404/ab166b.

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.

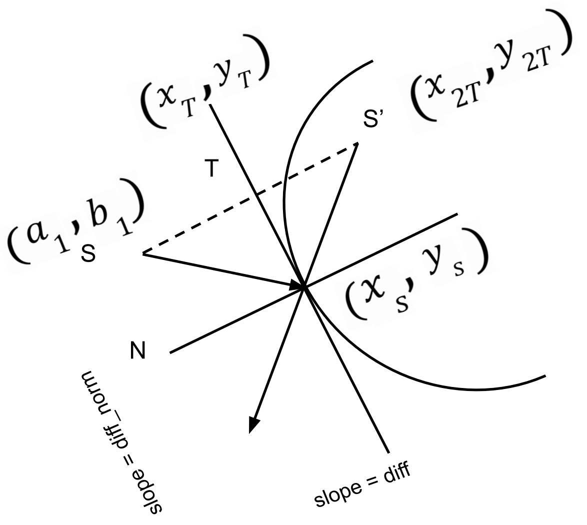


Figure 7: The Mirroring Technique

$$y_T = x_T.\text{diff} + c_1 \tag{18}$$

$$y_T = (x_T - a_1).\text{diff\_norm} + b_1 \tag{19}$$

By rearranging Eqs.(18) and (19), we obtain :

$$x_T = \frac{c_1 - b_1 + a_1.\text{diff\_norm}}{\text{diff\_norm} - \text{diff}} \tag{20}$$

Using the mid-point theorem of a line segment,

$$x_T = \frac{x_{2T} + a_1}{2} \tag{21}$$

$$y_T = \frac{y_{2T} + b_1}{2} \tag{22}$$

Rearranging (21) and (22), we obtain :

$$x_{2T} = 2x_T - a_1, y_{2T} = 2y_T - b_1 \tag{23}$$

Using (20) and (18), we obtain (x_{2T}, y_{2T}).

We also know that,

$$\frac{y - y_{2T}}{x - x_{2T}} = \frac{y_s - y_{2T}}{x_s - x_{2T}} \tag{24}$$

This gives us

$$y = \frac{(y_s - y_{2T})(x - x_{2T})}{x_s - x_{2T}} + y_{2T} \tag{25}$$

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 :

$$\frac{m_1 - m}{1 + m_1 m} = \frac{m - m_2}{1 + m_2 m} \tag{26}$$

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

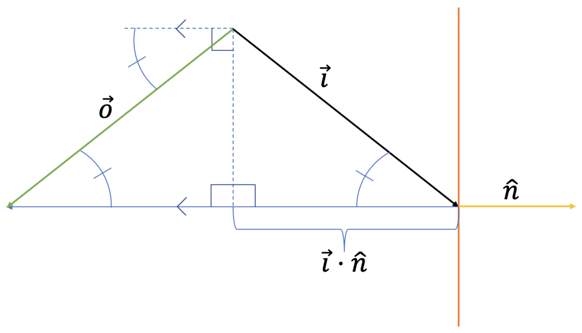


Figure 8: Vector Technique for finding the Reflected Ray

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

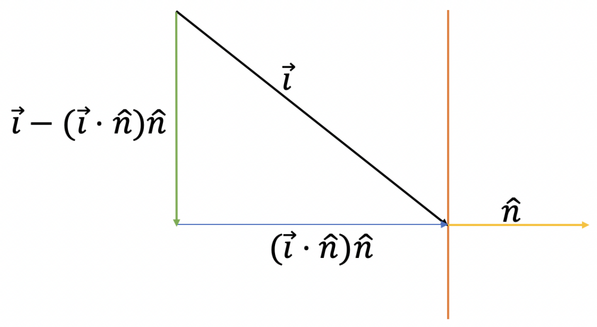

$\vec{\imath}$
$\vec{\imath} - (\vec{\imath} \cdot \hat{n})\hat{n}$
$\hat{n}$
$(\vec{\imath} \cdot \hat{n})\hat{n}$

Figure 9: Vector $\vec{i}$ derived from addition of orthogonal vectors

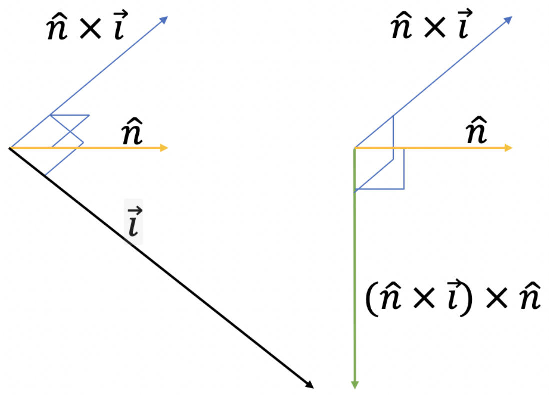

$\hat{n} \times \vec{\imath}$
$\hat{n}$
$\vec{\imath}$
$\hat{n} \times \vec{\imath}$
$\hat{n}$
$(\hat{n} \times \vec{\imath}) \times \hat{n}$

Figure 10: 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 9 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 10.

Also, in the Figure 10, 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,

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

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

$$n_1 sin\theta_1 = n_2 sin\theta_2 \tag{28}$$

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

$$\mu sin\theta_1 = sin\theta_2 \tag{29}$$

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

$$sin\theta = (A \times B)/\|A\|\|B\| \tag{30}$$

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

$$\mu(\hat{n} \times \hat{i}) = \hat{n} \times \vec{r} \tag{31}$$

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

$$\vec{r} = (\vec{r} \cdot \hat{n})\hat{n} + \vec{r} - (\vec{r} \cdot \hat{n})\hat{n} = (\vec{r} \cdot \hat{n})\hat{n} + (\hat{n} \times \vec{r}) \times \hat{n} \tag{32}$$

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

$$\vec{r} = (\vec{r} \cdot \hat{n})\hat{n} + \mu(\hat{n} \times \hat{i}) \times \hat{n} \tag{33}$$

The cross products can be replaced once again, giving us

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

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, $\hat{r}$.

$$\hat{r}^2 = ((\vec{r} \cdot \hat{n})\hat{n})^2 + (\mu(\hat{i} - (\hat{i} \cdot \hat{n})\hat{n}))^2 + 2(((\vec{r} \cdot \hat{n})\hat{n}) \cdot (\mu(\hat{i} - (\hat{i} \cdot \hat{n})\hat{n}))) \tag{35}$$

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

$$\hat{r}^2 = (\vec{r} \cdot \hat{n})^2\hat{n}^2 + \mu^2(\hat{i}^2 - 2(((\hat{i} \cdot \hat{n})\hat{n}) \cdot (\hat{i})) + (\hat{i} \cdot \hat{n})^2\hat{n}^2) \tag{36}$$

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

$$\hat{r}^2 = (\vec{r} \cdot \hat{n})^2 + \mu^2(1 - 2(\hat{i} \cdot \hat{n})2^{+}(\hat{i} \cdot \hat{n})^2) = (\vec{r} \cdot \hat{n})^2 + \mu^2(1 - (\hat{i} \cdot \hat{n})^2) \tag{37}$$

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

$$(\vec{r} \cdot \hat{n}) = \pm\sqrt{1 - \mu^2(1 - (\hat{i} \cdot \hat{n})^2)} \tag{38}$$

. 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

$$\vec{r} = \mu(\hat{i} - (\hat{n} \cdot \hat{i})\hat{n}) + \hat{n}\sqrt{1 - \mu^2(1 - (\hat{n} \cdot \hat{i})^2)} \tag{39}$$

E Algorithms

Algorithm 1 ConvertDecimalToBinaryUsingModF

```
Input: numberToConvert, numberOfDecimalPlaces = 15 ▷ Decimal number to Convert
Output: binaryNumber
 1: if numberToConvert < 0 then
 2:     sign ← 1
 3:     numberToConvert ← |numberToConvert|
 4: else
 5:     sign ← 0
 6: end if
 7: iIntegerPart ← int(numberToConvert)
 8: fDecimalPart ← numberToConvert - iIntegerPart
 9: iDecimalPart ← int(fDecimalPart * 10 ** numberOfDecimalPlaces)
10: binaryIntegerPart ← bin(iIntegerPart)
11: binaryDecimalPart ← bin(iDecimalPart)
12: leadingZeroes ← numberOfDecimalPlaces - length(iDecimalPart)
13: sLeadingZeroesDecimalPart ← "0" * leadingZeroes
14: return sign binaryIntegerPart . sLeadingZeroesDecimalPart binaryDecimalPart
```

Algorithm 2 ConvertBinaryToDecimalUsingModF

```
Input: $numberToConvert$                                  ▷ Binary number to Convert
Output: $decimalNumber$
 1: $binIntegerPart \leftarrow$ digits before the decimal point in $numberToConvert$
 2: $binDecimalPart \leftarrow$ digits after the decimal point in $numberToConvert$
 3: if $binaryIntegerPart[0] == 1$ then
 4:     $sign \leftarrow -$
 5: end if
 6: $binIntegerPart \leftarrow binIntegerPart[1:]$
 7: $iIntegerPart \leftarrow int(binIntegerPart, 2)$
 8: $iDecimalPart \leftarrow int(binDecimalPart, 2)$
 9: $binDecimalPartTemp \leftarrow bin(iDecimalPart)$
10: $leadingZeroes = \text{length}(binDecimalPart) - \text{length}(binDecimalPartTemp)$
11: prepend $0 * leadingZeroes$ to $iDecimalPart$
12: return $sign\ iIntegerPart.iDecimalPart$
```

Algorithm 3 ConvertBitstringToMatrix

```
Input: $bitString$
Output: $matrix,\ lastBit$
 1: if length($bitString$) in $lengthToMatrixSize$ then          ▷ Precomputed dictionary
 2:     $lastBit \leftarrow -1$                                   ▷ Not required
 3:     $RowCount \leftarrow lengthToMatrixSize.Rows$
 4:     $ColumnCount \leftarrow lengthToMatrixSize.Columns$
 5:     $tempRow \leftarrow 0$
 6:     $tempColumn \leftarrow 0$
 7:     for $i <$ length($bitString$) do
 8:         $matrix[tempRow][tempColumn] \leftarrow bitString[i]$
 9:         $tempColumn \leftarrow tempColumn + 1$
10:         if $tempColumn$ mod $ColumnCount$ then
11:             $tempColumn \leftarrow 0$
12:             $tempRow \leftarrow tempRow + 1$
13:         end if
14:     end for
15: else
16:     $bitString' \leftarrow bitString[0] \dots bitString[\text{length}(bitString) - 2]$
17:     $lastBit \leftarrow bitString[\text{length}(bitString) - 1]$
18:     $matrix \leftarrow ConvertBitstringToMatrix(bitString')$
19: end if
20: return $matrix, lastBit$
```

Algorithm 4 ConvertMatrixToBitstring

```
Input: matrix, lastBit
Output: bitString
1: bitString ← emptyString
2: for $i <$ RowCount(matrix) do
3:     for $j <$ ColumnCount(matrix) do
4:         Append matrix[i][j] to bitString
5:     end for
6: end for
7: if lastBit $\neq -1$ then
8:     Append lastBit to bitString
9: end if
10: return bitString
```

Algorithm 5 ApplyMatrixMix

```
Input: matrix, rowCount, columnCount, MatrixMix
Output: matrixMixed
1: matrixMixed ← emptyMatrix of size rowCount $\times$ columnCount
2: for $i <$ rowCount do
3:     for $j <$ columnCount do
4:         itemIndex ← MatrixMix[$(i, j)$]
5:         matrixMixed[i][j] ← matrix[itemIndex[0]][itemIndex[1]]
6:     end for
7: end for
8: return matrixMixed
```

F Table Summaries

Length	Size	Length	Size	Length	Size	Length	Size
4	(2, 2)	6	(2, 3)	8	(2, 4)	9	(3, 3)
10	(2, 5)	12	(3, 4)	14	(2, 7)	15	(3, 5)
16	(4, 4)	18	(3, 6)	20	(4, 5)	21	(3, 7)
22	(2, 11)	24	(4, 6)	25	(5, 5)	26	(2, 13)
27	(3, 9)	28	(4, 7)	30	(5, 6)	32	(4, 8)
33	(3, 11)	34	(2, 17)	35	(5, 7)	36	(6, 6)
38	(2, 19)	39	(3, 13)	40	(5, 8)	42	(6, 7)
44	(4, 11)	45	(5, 9)	46	(2, 23)	48	(6, 8)
49	(7, 7)	50	(5, 10)	51	(3, 17)	52	(4, 13)
54	(6, 9)	55	(5, 11)	56	(7, 8)	57	(3, 19)
58	(2, 29)	60	(5, 12)	62	(2, 31)	63	(7, 9)
64	(8, 8)						

Table 3: Bit-string length mapped to Matrix Size

G Hybrid Elliptic Curve Cryptography

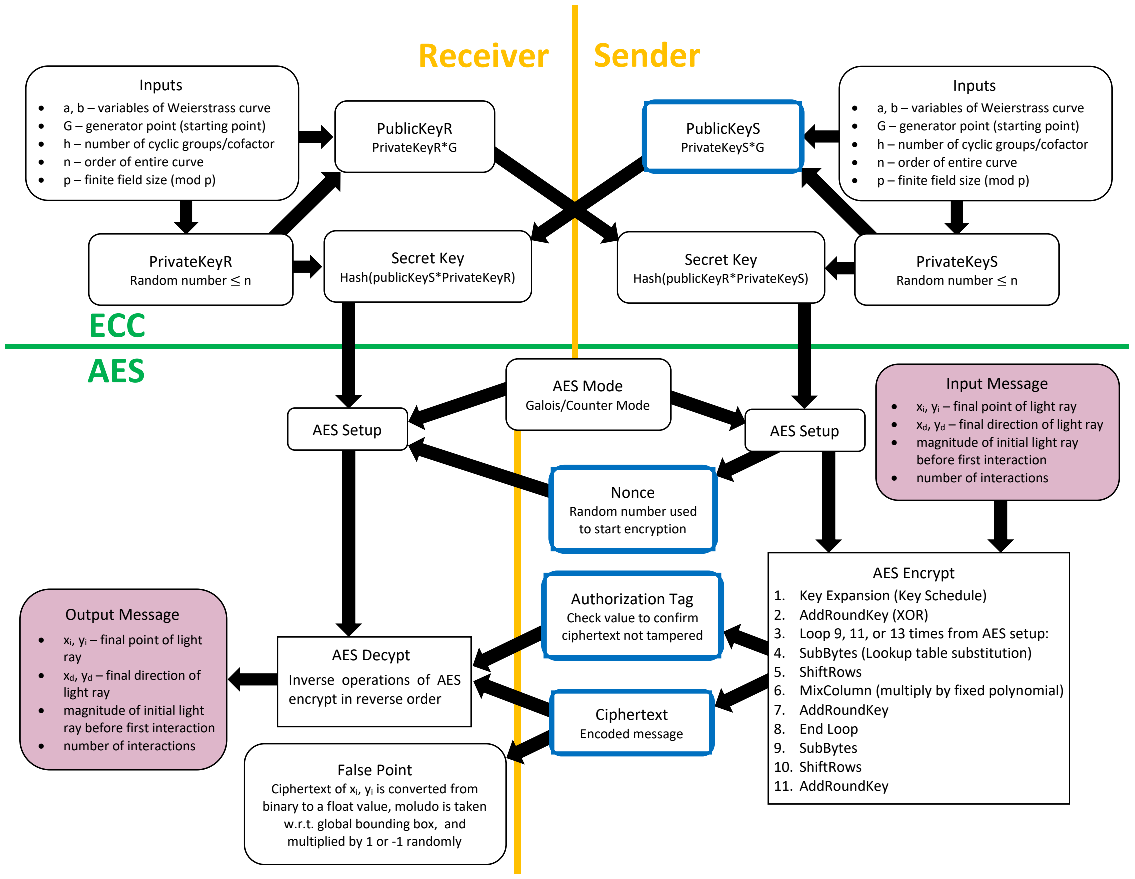

Receiver
Sender
Inputs
a, b – variables of Weierstrass curve
G – generator point (starting point)
h – number of cyclic groups/cofactor
n – order of entire curve
p – finite field size (mod p)
PublicKeyR
PrivateKeyR*G
PublicKeyS
PrivateKeyS*G
PrivateKeyR
Random number ≤ n
Secret Key
Hash(publicKeyS*PrivateKeyR)
Secret Key
Hash(publicKeyR*PrivateKeyS)
PrivateKeyS
Random number ≤ n
ECC
AES
AES Mode
Galois/Counter Mode
AES Setup
AES Setup
Input Message
xi, yi – final point of light ray
xd, yd – final direction of light ray
magnitude of initial light ray before first interaction
number of interactions
Nonce
Random number used to start encryption
AES Encrypt
1. Key Expansion (Key Schedule)
2. AddRoundKey (XOR)
3. Loop 9, 11, or 13 times from AES setup:
4. SubBytes (Lookup table substitution)
5. ShiftRows
6. MixColumn (multiply by fixed polynomial)
7. AddRoundKey
8. End Loop
9. SubBytes
10. ShiftRows
11. AddRoundKey
Authorization Tag
Check value to confirm ciphertext not tampered
Output Message
xi, yi – final point of light ray
xd, yd – final direction of light ray
magnitude of initial light ray before first interaction
number of interactions
AES Decypt
Inverse operations of AES encrypt in reverse order
Ciphertext
Encoded message
False Point
Ciphertext of xi, yi is converted from binary to a float value, moludo is taken w.r.t. global bounding box, and multiplied by 1 or -1 randomly

Figure 11: A flowchart illustrating the various components of the Hybrid ECC Cryptographic scheme.