RGBA Image Cryptography using Elliptic Curve RSA and Undeniable Digital Signature

Hokki Suwanda – 13519143[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
[1]13519143@std.stei.itb.ac.id

***Abstract*—Security is a very big issue in modern era where everything is transmitted through internet. There are a lot of possible attack happening on transmission process. Many methods and means have been used for securing transmission. A method is by applying cryptography, encryption and decryption, to the data transmitted.**

Keywords—security; transmission; encryption; descryption

I. Introduction

Everyone has some privacy that must not be known by others. Generally, every party has some privacy. Privacy is related to keeping secrets. However, privacy is not only about secrecy, but also ways of communication of multiple parties. Many private ways of communication exists, one of which is cryptography. Ironically, some cryptography algorithm require public involvement.

Prime numbers are a very important element in mathematics, especially discrete mathematics. Various researches and experiments about prime numbers have been conducted. Some are about identifying prime numbers, some are about primality testing, some are about the applications of prime numbers. Identifying prime numbers are done by primality testing. Primality testing algorithms available are not able to fully determine whether a number is prime. Because of that, prime numbers have a very big value. Even if the algorithm can correctly determine primality, like Sieve of Erathosthenes, they tend to be slow that they cannot be used to determine a very big prime number.

Prime number is a very fundamental element in number theory. More over, prime numbers are also a fundamental element in cryptography. All cryptographic algorithm needs a big prime numbers for security. That is why, bigger prime numbers has a very big value on 'securitians', a self-made term for people working in security. Needless to say, prime numbers are naturally used in cryptography as if cryptography breathes prime numbers.

Cryptography is classified into symmetric key cryptography and asymmetric key cryptography. Symmetric key cryptography, as its name suggests, is a cryptographic method that uses the same key for both encryption and decryption. Assymmetric key cryptography uses different keys for encryption and decryption. Symmetric key cryptography, because of its simplicity, is much easier and needs less time to implement compared to assymmetric key cryptography, with the drawback being less secure.

Key management has always been the problem for any cryptographic algorithm, especially symmetric key cryptography. Key management involves generating and distributing the key to stakeholders involved. Symmetric key cryptography uses same keys for both encryption and decryption, which means that the key must be distributed very carefully and securely. Key management is not a big problem in assymmetric cryptography because it uses two kinds of key.

Images are widely used on the internet. The security and privacy of images transmitted through the internet is a very important issue. Cryptograpy solves that issue. However, Not every cryptography is secure enough to secure images transmitted through the internet.

II. Theoretical Base

A. *Symmetric-Key Cryptography*

Symmetric-key cryptography is a system of cryptography where the encryption key is the same as the decryption key. Symmetric key is very versatile to brute force attack and cryptanalysis because of its simplicity. Symmetric-key cryptography itself is very limited as it mostly uses ASCII characters. One example of symmetric-key cryptography is Caesar Cipher.

Caesar cipher is an old cryptography algorithm based on alphabet. Caesar cipher uses substitution technique as its main principle. Substitution technique, as its name suggests, substitutes a letter of plain text with a letter of cipher text. Caesar cipher uses caesar wheel, which contains two layer of alphabet letters placed circularly in one direction.

B. *Public-Key Cryptography*

Public-key cryptography, which is also called assymmetric cryptography, uses different types of keys for encryption and decryption, unlike symmetric-key cryptography. As its name suggests, public-key cryptography involves publicly publishing the encryption. However, only the accomplice can make use of the key published and apply cryptography to communicate with each other. Public-key cryptography is often slower than symmetric-key cryptography because public-key cryptography is more complex.

There are many public-key cryptography algorithms available. Some examples are RSA, Blum-Goldwasser, Pohlig-Hellman, El-Gamal, and elliptic-curve-based algorithms. El-Gamal and RSA both uses prime numbers. The security of both El-Gamal and RSA is fully dependent to the size of the prime number used. Increasing the size of the prime numbers used will increase the security of the algorithm. However, it comes with a drawback of harder computations. Basic El-Gamal cryptography algorithm is as follows:

1) Generate a very big prime p and a defined number g. Both p and g are an agreement between accomplices

2) The receiver generates a random integer x as his/her private key and the corresponding public key (p, g, g^x) and publishes the public key

3) After receiving the public key, the sender splits the message to some blocks, each block can be interpreted as an integer m in interval $[0, p – 1]$

4) The sender generate a random integer k and send the encrypted message as (g^k, mg^{xk}) (mod p) to the receiver. Let it be at the form of (γ, δ)

5) The receiver uses his/her private key x by calculating the value of δ / γ^x.

Bigger p and g decreases the chance of the cipher being insecure. El-Gamal is inspired from discrete logarithm problem which is as follows "Given g^x (mod p), it is very difficult to calculate x if g and p" if both g and p are big.

C. RSA

RSA, Rivest-Shamir-Adleman, is found by three researchers from Massachussets Institute of Technology, Ronald Rivest, Adi Shamir, Leonard Adleman in 1976. Of all public-key algorithms proposed on the same era as or before RSA, RSA is the easiest to understand and implement. RSA uses the difficulty of factoring large number as its idea. Both the public key and private key are both functions of two very large prime numbers. This simple yet secure mechanism has proven itself by withstanding years of extensive cryptanalysis. Its key generation mechanism is as follows:

1) All accomplices generate two very big prime numbers p and q. Then calculate their product, n.

2) Receiver randomly chose the public key e such that e and $(p – 1)(q – 1)$ are relatively prime

3) Receiver then needs to calculate private key d using extended Euclidean Algorithm such that

$$ed \equiv 1 \bmod (p-1)(q-1) \tag{1}$$

4) Both p and q can be discarded.

To encrypt message m, the sender must first divide it to numerical blocks, with each block is smaller than n. The encrypted message, c, is made up of every encrypted block, c_i, where

$$c_i = m_i^e \bmod n. \tag{2}$$

To decrypt a message, take each encrypted block c_i and calculate

$$m_i = c_i^d \bmod n. \tag{3}$$

RSA is a slow algorithm as it operates on very big numbers. RSA encryption can go much faster depending on the value of e. For the value of e, PEM recommends 3, X.509 recommends 65537, PKCS recommends 3 or 65537.

D. Elliptic Curve

Elliptic curves are curves satisfying

$$y^2 = x^3 + ax + b \tag{4}$$

where

$$0 \neq 4a^3 + 27b^2. \tag{5}$$

Elliptic curve is a cause for elliptic curve discrete logarithm problem. Calculating $Q = kP$ given k and P is easy, but calculating k given P and Q is very difficult with both P and Q is a point on Elliptic curve, especially when k is big. In elliptic curve, kP is analogous to P^k, which has discrete logarithm problem.

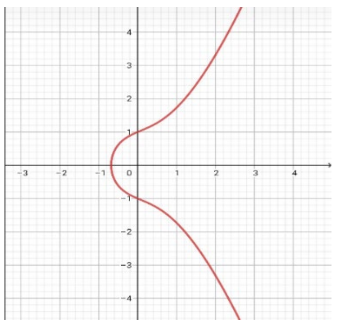


Fig. 1. $y^2 = x^3 + x + 1$ (GeoGebra)

Let $p > 3$ be a prime number. An elliptic curve $y^2 = x^3 + ax + b$ over Z_p is a set E of all (x, y) where $y^2 \equiv x^3 + ax + b$ (mod p). A particular point O is then added to E. O is called the point at infinity. The result r of additions, subtractions, and scalar multiplication on any element e in E will always be a subset of E.

Generally, elliptic curve supports two main operations:

1) Addition
For given $P(x_p, y_p)$ and $Q(x_q, y_q)$ where $P + Q = R(x_r, y_r)$, R is calculated depending on P and Q. If both of P and Q are two different points with the same absis, R is point at infinity. Other cases are calculated by:

$$x_r = m^2 - x_p - x_q \tag{6}$$

$$y_r = m(x_p - x_r) - y_p \tag{7}$$

$$m = (y_p - y_q)(x_p - x_q)^{-1} \bmod p,\ P \neq Q \tag{8}$$

$$m = (3x_p^2 + a)(2y_p)^{-1} \bmod p,\ P = Q \tag{9}$$

2) Multiplication
Multiplication is repeated addition. Only scalar multiplication is applicable to elliptic curves. Multiplication for $k = 2$ is often called doubles. Multiplicating point at infinity will result in point at infinity. Multiplicating a point by zero will result in point at infinity. Generally, for integer k, scalar multiplication of $P(x_p, y_p)$ on elliptic curve E with n points, including point at infinity, is as follows:

$$kP = (k + n)P = P + P + P + \ldots + P \text{ for } k \text{ times} \tag{10}$$

$$-P = (x_p, -y_p) \bmod p \tag{11}$$

E. Elliptic Curve RSA

Elliptic curve RSA is a variation of elliptic curve cryptography where elliptic curve is combined with RSA. The schemes are similar to that of RSA, with a few modifications made by the writer. For elliptic curve that has n points, including the point at infinity, and a point P, kP is recurring, which means that

$$kP = (k + n)\mathrm{P} \tag{12}$$

If RSA has two numbers e, d so that e and d satisfies (1), then ECRSA has two numbers e, d so that e and d satisfies

$$ed \equiv 1 \pmod{n}. \tag{13}$$

F. Hash Function

A hash function is a function that takes an input string and converts it to a fixed-length output string. The output string is called digest or hash value, while the input is called pre-image. Hash is a one-way function, which means that the process of hashing is irreversible. Any input string will result in the same length of hash value, irrelevant to length of the input. For this reason, a hash function can be considered as a compression function.

Because of one-way property that a hash function has, it is very difficult to generate a pre-image that corresponds to the given hash value. However, it is very easy to determine the hash value of a given pre-image. Hash values has some other properties: This template was designed for two affiliations.

1) Collision resistance
2) Pre-image resistance
3) Second pre-image resistance

Collision resistance means that it is very difficult to find two different pre-images with the same hash value. Pre-image resistance means that it is very difficult to find the pre-image given the hash value. In fact, it is computationally unfeasible to find the pre-image. Second pre-image resistance means that it is very difficult to find a different pre-image n such that the hash value is h, for h is a hash value of given pre-image m.

Several applications of one-way hash functions include message integrity, message compression, and normalization. Hash function is a volatile functions, a change in one bit will cause the hash value very significantly. With this, it means that when a hash value of a file is different than the official hash value, it is safe to say that the file is modified. However, it is very hard to prove that the file is not modified because some hash functions are not collision resistance. Some widely known hash functions are MD5, SHA-256, SHA-384, and SHA-512.

G. Digital Signature

One application of public-key cryptography for authentication service is digital signature. In signing the message, the sender uses his/her private key to encrypt the message. The signature is then embedded to the message, which are both sent to the receiver. In practice, the message is initially hashed to prevent the similarity of digital signature and the message. A digital signature is used to verify the sender of the message. If the sender of the message is not an accomplice, there are two possibilities: either the message is not signed or the signature is invalid. This is a property of digital signature where only an accomplice can correctly verify another accomplice.

The first method found was analogous to RSA, namely RSA signature scheme. RSA signature scheme is a recurring method. Other than RSA signature scheme, there exists other digital signature methods with the difference being in implementation and computation time. El-Gamal can also be used to digitally sign a message. There are also algorithms specialized in signing a message digitally like DSA and DSS.

According to Schneier, the characteristics of a digital signature scheme is:

1) Signature is authentic. The signature convinces the document's recipient that the signer deliberately signed the document.

2) Signature is unforgeable. Signature is a proof that the signer, and no one else, deliberately signed the document.

3) Signature is not reusable. The signature is part of the document, and unscrupulous person cannot move the signature to a different document.

4) Signature cannot be repudiated. The signature and the document are physical things. The signer cannot later claim that he/she did not sign it.

5) Signed documents are unmodifiable.

Let m be the message the sender wanted to send, using El-Gamal with his/her private key x and public key (p, g, y) where

$$y = g^x. \tag{14}$$

Sender then digitally signs m with few steps. The sender uses his/her private key and signs m with (γ, δ), where

$$\gamma \equiv y \tag{15}$$

and

$$\delta \equiv my^y. \tag{16}$$

The receiver can verify the signature by calculating δ / γ^y.

Digitally signing a message m with RSA is almost the same as encrypting a message, but instead of using public key, it uses private key to sign a message. Sender encrypts the message using his/her private key, d, which then the result is embedded to the message.

Generally, digital signature is used together with hash functions. General digital signature scheme starts with hashing the message m into h, then encrypts h by using any encryption method into c which is then embedded to the message.

H. Undeniable Digital Signature

Undeniable digital signature schemes are a little bit distinct from usual normal signatures in a sense that undeniable digital signature schemes requires the cooperation of the signer. A scheme available is Chaum-van Antwerpen algorithm, which the key generation scheme is similar to El-Gamal. Key generation scheme for signer in Chaum-van Antwerpen algorithm is as follows:

1) Generate a random prime $p = 2q + 1$ where q is also a prime

2) Select an integer α for the subgroup of order q in Z_p^*

3) Randomly generate private key a from $\{1, 2, ..., q - 1\}$ and compute $y = \alpha^a \bmod p$

4) Publicly publish (p, α, y) as the signer's public key

The form of public key is similar to that of El-Gamal's. For example, if A signed a message m belonging to the subgroup of order q, B can verify this signature with the cooperation of A. In the process of signing m, A can sign the message by calculating the sign $s = m^a \bmod p$. To verify the signature, the protocol is:

- B randomly choose two random secret integers γ, δ from $\{1, 2, ..., q - 1\}$
- B computes $z = s^\gamma y^\delta \bmod p$ and sends z to A
- A computes $i \equiv a^{-1} \bmod q$ and computes $w = z^i$ and sends w to B
- B computes $w' = m^\gamma \alpha^\delta \bmod p$

The signature is verified if and only if $w = w'$. The proof is as follows:

$$w \equiv z^i \equiv (s^\gamma y^\delta)^i \equiv (m^{\gamma a} \alpha^{\delta a})^i \equiv m^\gamma \alpha^\delta \equiv w' \pmod{p} \tag{17}$$

III. Implementation

A. Encoding

Every images are handled in four channels, RGBA, which is handled as three dimensional array of size *height* x *width* x 4 using python programming language. Every possible values in RGBA, namely [0, 255], is encoded to different points in a certain elliptic curve and every point must be decoded to different RGBA values. Because of this encoding system, an elliptic curve with exactly 256 points are needed. However, point at infinity has to be taken into account in the 256 points stated previously. The elliptic curve chosen is:

$$y^2 = x^3 + x + 1 \pmod{277} \tag{18}$$

or an elliptic curve with parameters $a = 1$, $b = 1$, and $p = 277$.

The encoding table is saved as a 1-dimensional array that consists of 256 elements indexed from 0 to 255. The element at i is the point i is encoded into. Thus, decoding a point to RGBA value can be done by using the encoding table. If a point P is at index i in the encoding table, then P is decoded to i.

B. Encrypting Image

Encrypting an image is a continuation of encrypting a message. In encrypting a message using elliptic curve variations, we encodes one or some characters to a point in the defined elliptic curve, then encrypts the point that results in another point which is then decoded into characters. Because image is made of pixels of color compositions, images can be encrypted by manipulating the color compositions.

If cryptography is implemented on every color compositions, there will be *height* x *width* x 4 operations done on the image. Because of this behavior, encrypting the image takes very much computation time and operations. To optimize the encryption process, grouping pixels to blocks may be necessary but may cause the encryption to be lossy depending on the encoding method chosen. Grouping pixels to blocks can reduce computation time and operations drastically. To encrypt an image, the sender and receiver must define the elliptic curve used and their keys.

Key generation is done simultaneously for both the encryption process, signing process, decryption process, and verifying process. Generating the encryption and decryption key follows the following pseudocode:

```
function   generate_key(elliptic_curve)   returns
(int, int)
begin
   // number of points in an elliptic
   // curve, including point at infinity
   n = elliptic_curve.points.length

   // e is a prime between 0 and n
   e = random_prime(0, n)
   d = mod_inverse(e, n)
   return (e, d)
end
```

To encrypt an image, a pseudocode is provided below.

```
procedure encrypt_image(filename, key)
begin
   channel = "RGBA"

   image = read_image(filename, channel)
   width = image.width
   height = image.height
   img = image.asarray()
   for i = 0 to height - 1
      for j = 0 to width - 1
         for k = 0 to channel.length - 1
            img[i,j,k] = encrypt(img[i,j,k], key)

   // produce the resulting cipher image
   output_image = to_image(img)
   output_image.save('out/' + filename)
end
```

The algorithm first reads the image in RGBA. Having JPEG or JPG extension will not make the algorithm work because JPEG and JPG compresses the image, which uses lossy compression that can cause errors and changes in RGBA values. The algorithm checks for every pixel in the image, where each pixel contains four values between 0 and 255 inclusively. Each values are encrypted using the key provided.

The encryption using ECRSA uses only public key *e* with defined elliptic curve parameters in (18). The encryption algorithm for every RGBA value is simple, which is shown in the following pseudocode.

```
// defined elliptic curve here

function encrypt(value, key) returns int
begin
   // d is unused
   (e, d) = key
   // encode → encodes a value between [0, 255]
   // to a point in elliptic curve,
   // point at infinity included
   ecc_point = encode(value)

   // multiplying by a scalar in elliptic curve
   // is analogous to powering in numbers
   encrypted_point = ecc_point * e

   // decode the resulting point back to RGB
   // values
   return decode(encrypted_point)
end
```

C. *Signing Image*

Key generation process for signing and verifying a signature is as follows:

```
function  generate_signature_key()  returns  ((int,
int, int), int)
begin
   q = random_prime(64, 128)
   p = 2 * q + 1
   while not isprime(p) begin
      q = random_prime(64, 128)
      p = 2 * q + 1
   end
   alpha = random(2, q - 1)
   a = random(1, q - 1)
   return ((p, alpha, pow(alpha, a)), a)
end
```

The key used for digital signature is small, with only one byte in size, because the powers done by the algorithm will be in mod *p*, must be in the range where RGBA values are valid. In this context, the signature is only embedded at the end of file, which does not get reflected in the image shown.

Signing process first started by reading every byte in the file. Every bytes is raised to the power of private signature key *a*. The signature is then embedded as SSSSSIIIIIGGGGGNNNNN***** followed by the signed bytes content. The SSSSSIIIIIGGGGGNNNNN***** is used to mark the beginning of a digital signature. Only one SSSSSIIIIIGGGGGNNNNN***** marked signature can exist in a file. The signing process will cause the file size to be doubled.

```
function sign(filename, public_key, private_key) returns boolean
begin
   (p, alpha, y) = public_key
   a = private_key

   // check if the file is signed
   // signing a signed file = sign failed
   if is_signed(filename):
        return false

   content = read_file(filename)
   content_length = content.length
   content_bytes = content.to_bytes

   for i = 0 to content_length – 1 begin
      content_bytes[i] = pow(content_bytes[i], a)
   end
   // append signature to file
   write_file('signed-' + filename, content)
   write_file('signed-' + filename, "SSSSSIIIIIGGGGGNNNNN*****")
   write_file('signed- + filename, content_bytes)
   // signature successful
   return true
end
```

D. *Verifying Digital Signature*

Signature verification is started by reading every byte in a file and checks for SSSSSIIIIIGGGGGNNNNN***** embedded in the file. If it is not found, the verification process failed because there is no signature to verify. If it is found, it reads the embedded signature that comes after the mark. On verifying the signature, interaction between verifier and signer must be done. Writer implements the interaction as if it is done without any interaction which is shown in pseudocode below.

```
function verify (filename, public_key, private_key) returns boolean
begin
   (p, alpha, y) = public_key
   // p = 2q + 1
   q = (p – 1) / 2
   a = private_key

   // check if the file is signed
   if not is_signed(filename):
        return false

   content = read_file(filename)
   // parse the signature from whole file content
   content, signature = parse(content)

   // verifier generates 2 secret random numbers
   gamma = random(1, q – 1)
   delta = random(1, q – 1)

   // verifier computes z
   z = pow(signature, gamma) * pow(y, delta)

   // signer computes w
   a_inv = mod_inverse(a, q)
   w = pow(z, a_inv)

   // verifier computes w' (read: w prime)
   w_prime = pow(content, gamma) * pow(alpha, delta)

   return w = w_prime
end
```

E. *Decrypting Image*

To decrypt an image, a pseudocode is provided below.

```
procedure decrypt_image(filename, key)
begin
   channel = "RGBA"

   image = read_image(filename, channel)
   width = image.width
   height = image.height
   img = image.asarray()
   for i = 0 to height - 1
      for j = 0 to width - 1
         for k = 0 to channel.length - 1
            img[i,j,k] = decrypt(img[i,j,k], key)

   // produce the resulting image
   output_image = to_image(img)
   output_image.save('out2/' + filename)
end
```

The algorithm first reads the image in RGBA. Having JPEG or JPG extension will not make the algorithm work because JPEG and JPG compresses the image, which uses lossy compression that can cause errors and changes in RGBA values. The algorithm checks for every pixel in the image, where each pixel contains four values between 0 and 255 inclusively. Each values are decrypted using the key provided.

The decryption using ECRSA uses only private key d with defined elliptic curve parameters in (18). The decryption algorithm for every RGBA value is simple, which is shown in the following pseudocode.

```
// defined elliptic curve here
function decrypt(value, key) returns int
begin
   // e is unused
   (e, d) = key

   ecc_point = encode(value)

   // multiplying by a scalar in elliptic curve
   // is analogous to powering in numbers
   decrypted_point = ecc_point * d

   // decode the resulting point back to RGB
   // values
   return decode(decrypted_point)
end
```

IV. Analysis

The specifications of the computer used in the simulation is Intel® Core™ i3-2370M CPU @ 2.40GHz, 16GB RAM, Python 3.8.10 on Windows 7 (64-bit). The elliptic curve used has exactly 256 points with parameter

$a = 1$;

$b = 2$;

$p = 277$.

The keys used are as follows:

$e = 179$;

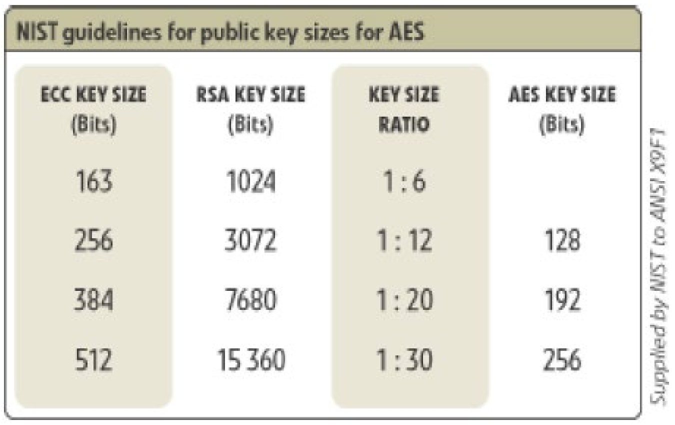
$d = 123$;

$(p_s, \alpha, y) = (179, 27, 66)$;

$a_s = 74$.

Where e is RSA encryption key, d is RSA decryption key, p_s is prime number used as a public key in undeniable digital signature, α is a prime number used as base in undeniable digital signature, y is a discrete exponential of α to a_s, the private key.

To encrypt or decrypt an AES key with 128 bits in size, key sizes varies depending on the algorithm used. For RSA, the key size needed is 3072 bits. For elliptic curve cryptography variations, only 256-bits keys are needed with a big ratio of 12 to 1. This proves that elliptic curve algorithm variations are much more secure than RSA with the same key size. One way to increase the security of RSA is by increasing the size of the keys. However, as key size increase, the number increases. Researches to identify big prime numbers are still being conducted, which means that RSA key size is still very limited.



NIST guidelines for public key sizes for AES

ECC KEY SIZE (Bits)	RSA KEY SIZE (Bits)	KEY SIZE RATIO	AES KEY SIZE (Bits)
163	1024	1 : 6	
256	3072	1 : 12	128
384	7680	1 : 20	192
512	15 360	1 : 30	256

Supplied by NIST to ANSI X9F1

Fig. 2. *NIST guidelines for public key sizes for AES* (*Rinaldi Munir*)

The algorithm first reads the image in four channels, which are R, G, B, and A that only exists in a PNG image file. The algorithm checks for every pixel in the image, where each pixel contains four values between 0 and 255 inclusively. Each values are then encrypted or decrypted using the key provided. The algorithm will check for *width* * *height* pixels. Thus, the algorithm is slow for images with very big size.

Moreover, as the key size used in encryption, decryption, signing, and verifying process are relatively small (8 or 9 bits), the security level is low. On the other hand, the key used will be relatively static because of the constraints applied, which makes the algorithm less secure.

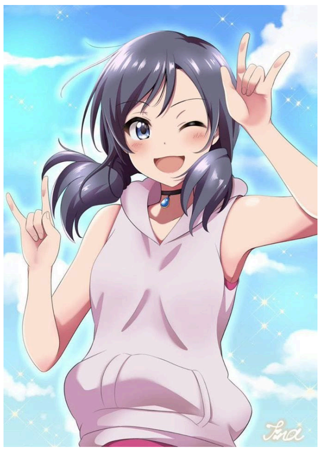


Fig. 3. True image

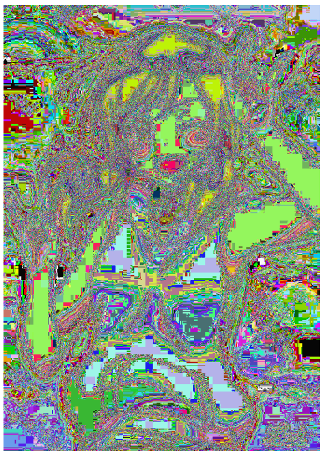


Fig. 4. Encrypted image

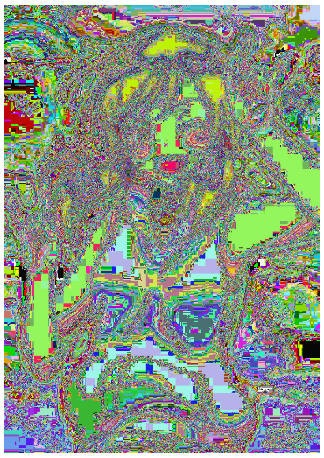


Fig. 5. Signed image

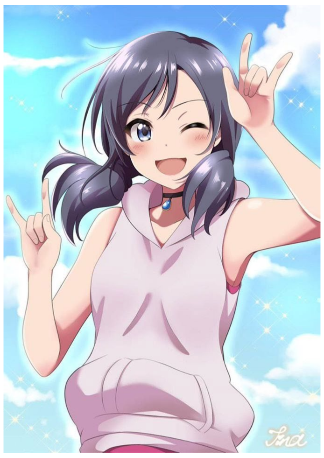


Fig. 6. Decrypted image

For images, visual cryptography is more recommended than normal cryptographic method because:

1) Visual cryptography uses less computation time and capacity

2) Decryption in visual cryptography can be done without any computations, as opposed to normal cryptographic method where it needs computations on big numbers

3) Visual cryptography does not need any keys for encryption and decryption, but uses share to denote the number of divisions done.

V. Conclusion

Cryptography has many uses, one of them is to secure image sent in the internet. There are many cryptographic algorithms available publicly. Some of them are El-Gamal, RSA, and elliptic curve algorithms. Elliptic curve algorithms are proven to be more secure than other cryptographic algorithms with the same key size. Security can be further increased by embedding a digital signature on the image. One kind of digital signature is undeniable digital signature that needs the involvement of the signer, thus the signer cannot deny that he/she signed the message.

Source code Link at Github

https://github.com/hokkyss/image-cryptography

Acknowledgment

The writer is grateful to Mr. Rinaldi Munir, the lecturer of IF4020 Cryptography who taught the writer on weekly lectures. The writer is also grateful to friends that listened to the writer when the writer has some problems regarding this paper. The writer is grateful to close friends who motivated and encouraged the writer when writing this paper.

References

[1] Menezes, Alfred J and friends. (1996). *Handbook of Applied Cryptography*. Massachusetts Institute of Technology.

[2] Scheneier, Bruce. (2015). *Applied Cryptography: Protocols, Algorithms, and Source Code in C.* Indianapolis: John Wiley & Sons.

[3] Jajodia, Sushil. (2011). *Encyclopedia of Cryptography and Security*. Springer Science+Business Media.

[4] Stallings, William. (2017). *Cryptography and Network Security. Principles and Practice. Seventh Edition Global Edition*. England: Pearson.

[5] Sumarkidjo, dkk. (2007). *Jelajah Kriptologi*. Jakarta: Lembaga Sandi Negara.

[6] Munir, Rinaldi. (2021). *Bahan Kuliah IF4020 Kriptografi*. Program Studi Informatika STEI-ITB.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Pekanbaru, 20 Desember 2021

[illegible]

Hokki Suwanda
13519143