# Implementation of a Data Protection Model dubbed Harricent_RSECC

Frimpong Twum[1], Vincent Amankona[2]*, Yaw Marfo Missah[3], Ussiph Najim[4], Michael Opoku[5]
Department of Computer Science, KNUST, Kumasi, Ghana[1, 3, 4]
Department of Computer Science, CUCG, Sunyani, Ghana[2]
Department of Computer Science, UENR, Kumasi, Ghana[5]

***Abstract*—Every organization subsists on data, which is a quintessential resource. Quite a number of studies have been carried out relative to procedures that can be deployed to enhance data protection. However, available literature indicates most authors have focused on either encryption or encoding schemes to provide data security. The ability to integrate these techniques and leverage on their strengths to achieve a robust data protection is the pivot of this study. As a result, a data protection model, dubbed Harricent_RSECC has been designed and implemented to achieve the study's objective through the utilization of Elliptic Curve Cryptography (ECC) and Reed Solomon (RS) codes. The model consists of five components, namely: message identification, generator module, data encoding, data encryption and data signature. The result is the generation of the Reed Solomon codewords; cipher texts; and generated hash values which are utilized to detect and correct corrupt data; obfuscates data; and sign data respectively, during transmission or storage. The contribution of this paper is the ability to combine encoding and encryption schemes to enhance data protection to ensure confidentiality, authenticity, integrity, and non-repudiation.**

***Keywords—Elliptic curve cryptography; encoding; encryption; Reed Solomon; security***

## I. Introduction

### A. Background of Study

The advent of computerized systems and networks has been beneficial to organizations and has subsequently enhanced their operations. This has resulted in the generation of larger quantum of data to augment the activities of these organizations. Data produced by these organizations are considered a major resource, therefore resulting in organizations adopting strategies that can protect this important resource from being misused. As a quintessential resource, comprehensive techniques are provided and instituted by these organizations frequently to offer protection to this data [1].

Though protective mechanisms are instituted, there is also an increase in threats to undermine organizations' operations. Notwithstanding, the insurgence of adversary's attacks have consistently hampered the functional activities of organizations over the past years and was colossal during the COVID-19 pandemic era. To ameliorate this insurgence, organizations started scrambling for solutions to protect their data. In this regard, researchers began to also seek for solutions to mitigate this insurgence of threats and attacks through the development of robust techniques and methods to prevent loss of data and unauthorized access and modification. It is on the basis of the aforementioned, that it is always important for industry and researchers to stay a step ahead of attackers in the preservation of data in transit and storage, hence, the call for this research.

As several research have postulated, most security systems have focused on either using encoding or encryption strategies/techniques to guarantee the safety and accuracy of data [2]. Whereas the encoding schemes add extra bits to the original data to aid in error detection and correction, in order to maintain the integrity of messages. The encryption schemes, on the other hand, ensure messages transmitted or stored are obfuscated to prevent unauthorized access and modification in order to maintain the authenticity and confidentiality of messages.

Examples of data encoding and encryption schemes include Reed Solomon codes, Reed-Muller codes, checksum, and AES, RSA, ECC, Blowfish among others have emerged to offer protection to data [3]. The encoding and encryption schemes are aimed at preserving the confidentiality, authenticity, integrity and non-repudiation (CAIN) of data. The utilization of encoding scheme over encryption algorithms, even though, offers security but it is ineffective to provide optimal protection due to unauthorized access or alteration of data as a consequence of adversaries' activities. This illicit access and modification of the data by the attacker causes data compromise. Also, the utilization of encryption algorithms offer protection to data but it is inadequate as a result of inadvertent modification, loss of data or hardware failures. Data can be destroyed by hardware failures, un-trusted communication channels or attackers when accessed. This also leads to data compromise and therefore requires the application of different security measures to offer optimal security.

This research therefore focuses on how to harness the strengths of encoding and encryption security techniques to achieve a robust data protection system.

### B. Aim of Study

The study's aim is to implement a data protection model by integrating Elliptic Curve Cryptography (ECC) and Reed Solomon (RS) coding schemes.

*Corresponding Author.

To achieve the stated aim the following questions are raised:

*1)* Can ECC and RS codes be used to ensure secure data transmission and secure data storage?

*2)* Can a proposed Harricent_RSECC data protection model enhance data security by ensuring an uncompromised data transmission and data storage?

*3)* Will the implementation of the proposed Harricent_RSECC data protection model offers the required security to data?

## II. Elliptic Curve Cryptography (ECC)

Elliptic Curve Cryptography (ECC) is a contemporary group of public-key cryptosystems premised on the algebraic structures of elliptic curves over finite fields and the complexity of the Elliptic Curve Discrete Logarithm Problem (ECDLP).

An ECC curve can be illustrated as a curve that intersects two lines on a graph. This type of curve is determined by the properties of the mathematical group consisting of set of values for which operation on two of its members produces a third member [4] depicted in Fig. 1. Multiplying a point by a number on the curve produces an additional point on the curve, but finding what number has been used is very difficult, although parties involved know the original point and results. ECC uses elliptic curves in which elements of a finite field are all limited to variables and coefficients. The ECC is mathematically represented using the Weierstrass form of an elliptic curve denoted in (1) as follows $y^2 = x^3 + ax + b$ where $4a^3 + 27b^2 = 0$. Each "a" and "b" value has an elliptic curve that is different.

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

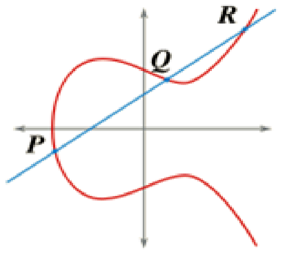


Fig. 1. ECC Representation.

ECC constructs all of the substantial functionalities of asymmetric cryptosystem, including encryption, signatures, and key exchange. ECC cryptography is regarded as an effective modern replacement to the RSA cryptosystem since it utilizes smaller keys and signatures than RSA for the same level of security and offers extremely fast key generation, key agreement, and signatures.

### A. ECC Keys, Algorithms, Curves and Key Length

In the ECC, the composition of private keys are generally numbers within a range of integers (usually 256-bit integers), thereby making it easier and faster for private key generation. The public key on the other hand is generated from points which lay on an elliptic curve, usually a pair of integer coordinates (x, y). Ultimately, a shared key is a public key that is derived after multiplying the private key of the sender to the public key of the receiver and vice versa [5].

Besides, basing on the mathematical properties of elliptic curves over finite fields, the elliptical cryptography offers varied sets of algorithms. Three major categories of ECC algorithms are available consisting: signature algorithms such as elliptic curve digital signature algorithms, fast elliptic curve digital signature algorithms and Edwards digital signature algorithms; encryption algorithms for instance elliptic curve integrated encryption scheme and ElGamal Encryption using ECC (EEECC); and lastly is the key agreement algorithms including elliptic curve Diffie Hellman X25519 and Fully Hashed Menezes-Qu-Vanstone (FHMQV) [6].

Moreover, diverse sets of curves exist that elliptic curve crypto algorithms can utilize to achieve different purposes. Goals such as determining the level of cryptographic strength (security), the length of key and the performance are rational for implementing a different set of curves. Consequently, every curve consists of the following parameters (curve name; size of the field/key; the cryptographic strength – expressed as ratio of field size to 2; and the speed – also expressed as ratio of operations to seconds). Examples of ECC curves and their corresponding key sizes include “curve secp192r1 with 192-bit, curves secp256k1 and Curve25519 with 256-bit, curve P-521 with 521-bit among others” [6].

Therefore, it can be advanced that digital signature, encryption and key agreement algorithms utilizes any of the curves above for computations which heavily rely on the difficulty of ECDLP to provide adequate level of security to data while performing with minimal key length.

### B. Key Benefits of Elliptic Curve Cryptography

The benefits of using elliptic curves stems from the fact that with shorter keys, equivalent levels of protection can be obtained compared with other algorithms. The ECC cryptosystem is useful in our contemporaries with massive upsurge in the development and usage of mobile devices. Thus, as the use of smartphones grows, more robust encryption is necessary for businesses to meet the increasing security requirements [7]. Some of the benefits are discussed below:

*1) Stronger keys:* ECC represents the latest encryption method, providing more security in elliptical curve cryptography. The underlying math problem of the ECC algorithm implies it is more difficult for hackers to crack compared to RSA and DSA, which makes the ECC algorithm more reliable than conventional methods for websites and infrastructure [8]. ECC algorithms rely on the difficulty of ECDLP to provide adequate level of security to data while utilizing minimal key length [5]. This invariably has improved performance and enhanced storage requirements.

*2) Short key size:* Comparably, Because ECC keys are substantially smaller than DSA/RSA keys; the size of the public and private keys in ECC is relatively short. As a result, the National Institute of Science and Technology (NIST) recommended that ECC can employ substantially lower parameters for the same degree of security bits as RSA/DSA [9]. The choice for these algorithms is that elliptic cryptography uses lesser keys to achieve same security levels contrary to the other PKCs. For instance, the RSA/DSA technique requires a key length of 7680 bits to achieve 192 bits of security, whereas ECC requires a key size of 384 bits. Furthermore, the RSA/DSA algorithm requires a key size of 15360 bits to obtain 256 bits of data protection, whereas ECC requires a key length of 512 bits.

As a result, processing times are reduced while memory and bandwidth requirements are limited. ECC is especially suitable for applications with limited memory, bandwidth, and/or computing capability, and its use is expected to increase in this area.

## III. Reed Solomon (RS) Coding Scheme

### A. *The Theory of Reed Solomon (RS) Coding*

The RS code is a systematic, linear cyclic and non-binary block code. During RS coding, redundant symbols are created and appended to the symbols of the message by using a polynomial generator [10]. The position and magnitude of the error in the decoder are determined using the same polynomial generator [11]. The correction is then applied to the code received. RS coding is commonly used for error detection in a variety of communications and computer infrastructures, including storage, wireless or mobile communications, satellite communications, digital television, and high-speed modems [12].

RS codes were opted over other error detection and correction codes because of their faster decoding capabilities, i.e., their ability to detect and/or correct significant numbers of omitted or compromised data items; and the fact that they require the fewest additional error correcting codes bits for a known number of data bits [10]. Fig. 2 provides a framework representation of RS encoding/decoding process:

Reed-Solomon codes are by far the most extensively utilized for burst error correction [13]. The benefit of utilizing RS codes is that the likelihood of an error persisting in the decoded data is (generally) substantially lesser than if RS codes are not employed. Coding gain is a common term for this benefit. Because Reed Solomon code has a high coding rate, it is suited for a broad array of applications, comprising storage and transmission [12].

### B. *Properties of RS Code*

RS code is specified as $RS(n, k)$ with $s$-bit symbols [10]. Given a symbol size s, the maximum codeword length (n) for a Reed-Solomon code is n = $2^s$ – 1. For instance, an 8-bit symbol will produce a maximum length of 255 bytes. This signifies that the encoder creates a $n$ symbol codeword by adding parity symbols to $k$ data symbols of $s$ bits each. There is a total of $n - k$ parity symbols, each with s bits. A Reed-Solomon decoder can fix up to $t$ symbols in a codeword that have errors, where $2t = n - k$. A typical Reed-Solomon codeword (known as a Systematic code because the data is not altered and the parity symbols are attached) is shown in Fig. 3 [14].

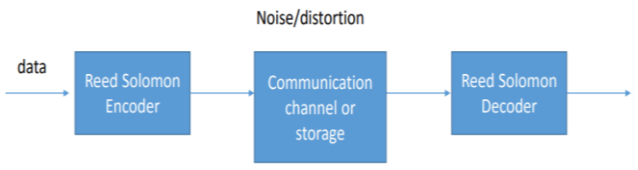


Fig. 2. Framework of RS Scheme.

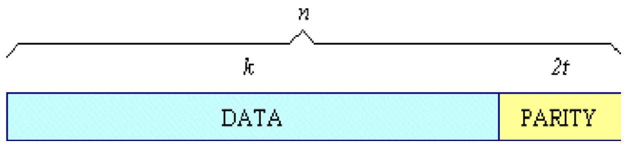


Fig. 3. Reed Solomon Codeword Generation.

A code such as Fig. 3 can detect and correct up to $(n - k)/2$ *or* $t$ symbol where each symbol represents an element within the finite fields $GF(2^m)$. This implies that any $t$ symbols that may be corrupted can still be recovered from the original message [15].

An RS(255, 223) with 8-bit symbols is an example. Each codeword is made up of 255 bytes, with 223 bytes of data and 32 bytes of parity. The following can be derived from the code:

n = 255; k = 223; s = 8; 2t = 32, t = 16

As a result, the decoder can automatically correct any 16 symbol errors in the codeword: that is, errors of up to 16 bytes can occur anywhere in the codeword.

In Fig. 4, the sender uses the RS encoder to encode the message in a codeword and transmits it through the communication channel. Channel noise and other disorders may disrupt the codeword and corrupt it. This corrupted codeword comes to the recipient end (decoder) and transfers the tested message to the receiver. If the error caused by the channel is larger than the decoder's error correction capability it may result in a decode failure. Decoding errors occur when a codeword has not been passed and a decoding error leads to an incorrect message [16].

The representation of an instance of an RS protected communication channel while transferring data is illustrated in Fig. 4.

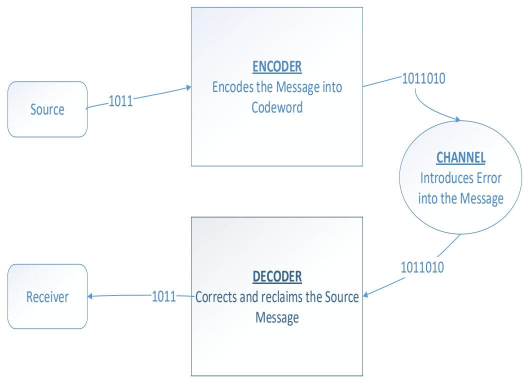

ENCODER
Encodes the Message into Codeword
Source
1011
1011010
CHANNEL
Introduces Error into the Message
1011010
DECODER
Corrects and reclaims the Source Message
1011
Receiver

Fig. 4. Reed Solomon Protected Channel.

## IV. Review of Related Work

This section explores the extant works of ECC and RS. Elliptic curve cryptography (ECC) is by far the most effective public-key option for providing security services to devices with limited resources and it has been employed in a variety of business applications [5]. ECC, since its emergence has been considered the preferred option with notable efficiency for ensuring authenticity, encryption, signatures and key agreements [5]. To achieve these, He et al. deployed an ECC authentication model in a smart grid environment to obfuscate smart meter anonymity [17]. However, Sadhukhan et al pointed some security flaws in [17] such as internal and masquerading attacks [9]. Besides, by utilizing ECC and image steganography to maintain the legality and accuracy of medical health data, Eshraq et al.'s model obfuscated health data from being accessed by unauthorized access [18]. To ensure ECC's applicability in diverse contexts, the authors [19], [20], [21], [22] and [23] utilized ECC to achieve confidentiality, integrity, non-repudiation and authenticity levels of security.

On the contrary, the establishment of a reliable communication channel is very necessary as there exist security breaches within satellite or telephone channel which can compromise data security [10]. To overcome these security breaches many techniques of correcting errors have been introduced over time. One of such method is RS code which is a key non-binary BCH coding sub-class. These are useful cyclic codes utilized in detecting and correcting burst errors. RS codes have been prevalent due to its simplicity in encoding and well-structured decoding capabilities. In appraising the efficiency of RS codes, Wonshik & Jae-Yeon appraised the performance of RS(255, 239) in a smart transmission medium and an intelligent system [12]. The implementation of their system depicted RS code efficiency over a chaotic communication channel as the higher the codeword length, the greater the bit error rate improvements. Besides, Mounika [10] postulated that there exist security breaches on a satellite or telephone channel. Therefore, the establishment of a reliable communication channel is very necessary. To this end, [10] implemented a modified version RS codes for performing error corrections. Further, studies by [13] indicated that the decoding capabilities of RS codes have been efficient against deletion errors. And it was validated through the simulation results by [24] where the proposed RS decoder achieved a higher coding gain in contrast to algebraic decoding methods.

The identifiable gap is the ability to combine both techniques to achieve a robust security model and that's the goal of this study.

## V. Implementation of the Harricent_RSECC Model

### A. Conceptual Framework of an Efficient Data Security

Quite a number of studies have been carried out relative to mechanisms that can be deployed to enhance data protection. However, most of these studies have either focused on encryption or encoding schemes as postulated in [2]. The ability to integrate these two techniques and leverage on their strengths to achieve adequate data protection has been the major concern in this study. To this end, the Harricent_RSECC data protection model has been designed and implemented to achieve the objective of this study through the utilization of Elliptic Curve Cryptography (ECC) and Reed Solomon (RS) codes. The prime goal for integrating ECC and RS is to achieve a fast, small, and portable cryptographic protocol, which would support elliptic curve digital signature generation and verification together with data reconstruction.

Fig. 5 shows the design of the Harricent_RSECC data protection paradigm, which integrates RS encoding and two variants of ECC to serve a purpose of improving data protection while achieving data integrity and confidentiality. Generally, the implementation of the model is achieved by the following steps.

### B. Metrics used for Harricent_RSECC Data Protection Model

As a defensive system, it is always important to identify any messages or data received. Since messages come in different forms, understanding the different message types received play a crucial role when identifying the message/data received. The process for an unwanted message to be detected or identified is important to ensure the right defensive mechanisms are applied.

Having identified the type of message, a second defense module, the generator module, ensures proper defense by identifying which encoding level is crucial in identifying and correcting the right amount of errors that may occur in the system. This module also provides the chunking of data to ensure faster processing of the messages received.

Thereafter, messages received are transferred for security mechanisms to be applied to the message. The mechanisms are to encode, encrypt and sign the messages. These three effective security mechanisms will prevent unauthorized access to the message received.

As different attacks can be performed, each stage in the framework provides a defensive approach in protecting the message received.

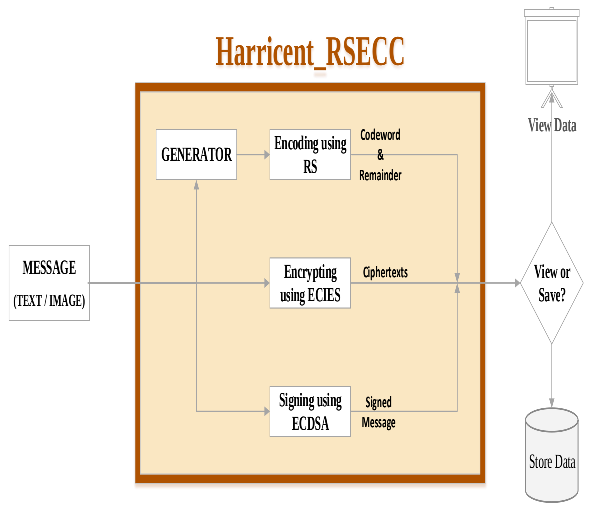


Fig. 5. Schematic Diagram of Harricent_RSECC Data Protection Model.

Therefore, the metrics that defined the functional requirements for Harricent_RSECC are:

*1)* The model must be able to identify the message type in order to perform the proper analysis. This maintains the scope of the study (text and images) prevent unclassified messages from being uploaded.

*2)* The model must be able to identify the message size/length in order to apply appropriate RS coder levels.

*3)* The model should be able to detect if a message has been compromised and correct eventually.

*4)* The model should be able to obfuscate the message and prevent unsolicited access and modification.

*5)* The model should be able to maintain the integrity of the message and associate messages transmitted or stored with the sender (i.e., ensuring non-repudiation).

The primary focus of this model is the ability for the system to identify a message type received, chunking of large messages for faster processing, obfuscation of the message, signing of the message, detection and correction of compromised messages. This work attempts to provide a protective mechanism which organizations can rely on to protect their resources.

## *C. Implementation of Harricent RSECC Components*

From the design of the model, the following principal components are implemented: Message Identification, Generator Module, Encoding, Encryption, and Signature.

*1) Message identification:* A message is submitted by the user into the Harricent_RSECC model. This phase is to identify the type of message that is permissible to be uploaded into the model, be it a text or an image. This module consists of a rule-based component where rules are defined based on the list of file types and type of file uploaded or sent by the user.

For example, a typical rule might be if(filetype==filetypeslist) return true. The module identifies the message/data based on the type of file received/uploaded. The received message/data is processed and categorized to compute a value Mt based on two outputs, text (Dt) and Image (Di). Messages are identified based on f(Mt ) as follows:

$$f(M_t) = \begin{cases} D_t, & if(M_t = filetypeslist_t \wedge filetype = filetypeslist), \\ D_i, & if(M_t = filetypeslist_i \wedge filetype = filetypeslist). \end{cases}$$

If new message types are identified, there is the need for it to be added to the file type list to improve the efficiency of the system. If new message types are identified, there is the need for it to be added to the file type list to improve the efficiency of the system.

Files uploaded are checked by a function to prevent unclassified files from being uploaded.

```
#function file upload
if (filetype = ".txt" or filetype = ".doc" or filetype = ".docx" or filetype = ".png" or filetype = ".jpg" or filetype = ".jpeg" or filetype = ".gif"):
upload file
else
print("file type not supported")
```

*2) The Generator Module (GM):* This module prepares image and text files for the RS encoding scheme to be applied. It determines the length of message, chunks message based on encoder level and generates ASCII codes from message. The received message could be preprocessed by this module to determine the message's length. Message length, $M_L$ , determination effectively enables how the message can be encoded. If a message, M, is presented as a number of characters $w_1, w_2, \ldots, w_n$, then $M_L$ is determined by $w_n$, where n is the last returning value, resulting in $M_L = n$. This module also takes into accounts the coder level during processing to determine the bits-type needed during encoding. The size of the bits-type is determined based on the message length $M_L$. The bits size, k, of a message is the bits-type to be used during RS encoding. Selection of the bits-type for a message is determined based on

$$f(M_k) = \begin{cases} 4bits, & if(M_L < \alpha), \\ 8bits, & if(M_L \geq \alpha) \end{cases}$$

where α is a pre-defined threshold value of thirteen characters.

The algorithm for determining the length of text messaging is presented below.

```
#def determine the length of message.
Take the length of message using the length library.
#def set bit size.
 if messagelength < 13 and Bit_size=="8-bits":
Display a message that coder level must be either 'RS(15, 11) and RS(15, 9).
Set bit_size to 4-bits.
Else.
Pass
```

After the message length has been determined, a decision has to be made where the received message will be chunked (1) or not-chunked (0), utilizing the encoder level. The encoder level utilizes message length CML and codeword length CWL in its operation. All encoder levels are in the format: CL($CW_L$, $CM_L$), where CL is for coder level. Chunked messages $M_c$ are determined based on.

$$f(M_c) = \begin{cases} 0, & if(M_L \leq \mathrm{CM}_L), \\ 1, & if(M_L > \mathrm{CM}_L) \end{cases}$$

The algorithm for chunking a message is presented below.

```
#code for chunking data.

#def chunkmydata(self, string, n):

chunks = [string[i:i + n] for i in range(0, len(string), n)].

return chunks.
```

The last operation of this module is the conversion of a message to its ASCII format. Each character (C) in the message is converted to its *ASCII code* (AC) (ie. $C_1$=>$AC_1$,…,$C_n$=>$AC_n$).

The algorithm for messaging to their equivalent ASCII is presented below.

```
#def convert message length to ASCII codes.

letters = take the ordinal of the message characters to give
their ASCII Codes.
```

*3) Data encoding:* This module utilizes the RS error correcting code to encode messages prepared from the GM to generate codeword. This is to guarantee that messages in transit or storage are devoid of corruption or contain errors by appending redundant data to it. The framework employed multiple encoder levels to encode. The RS code is built on finite fields, which have the feature that any computation on field elements always returns an element in the field-set [14].

To generate an RS codeword, a polynomial generator $g(x)$ is used with the following notations: information block $i(x)$, codeword $c(x)$ and a primitive element $(\alpha)$ of the field. A polynomial generator is illustrated in (2) below:

$$g(x) = (x - \alpha^i)(x - \alpha^{i+1}) \dots (x - \alpha^{i+2t}) \qquad (2)$$

with the codeword generated as follows (3):

$$c(x) = g(x)i(x) \qquad (3)$$

Following steps are employed in the generation of codewords and remainders:

*a)* An encoder level (RS) must be selected. To encode, Reed Solomon coder makes use of different encoder levels. Eight (8) RSCoder levels (RS) were selected ranging from (15, 9) to (255, 251) as follows:

RSCoder(15, 9), RSCoder(15, 11), RSCoder(53, 37), RSCoder(255, 223), RSCoder(255, 239), RSCoder(255, 251), RSCoder(255, 247), RSCoder(255, 191).

The above RS encoder levels determine the message ($RS_M$) and codeword ($RS_C$) lengths.

*b)* Message (M) length is determined based on the encoder level (RS) selected.

Message length = len(Message)

*c)* If message (M) length is less than or equal to the message ($RS_M$) length of the encoder, message (M) is encoded and a remainder (R) is generated.

*d)* A message (M) length that exceeds the encoder level message length ($RS_M$), message (M) is chunked based on the message ($RS_M$) length of the encoder level using a chunk function explained above. Each chunked message is encoded and a remainder (R) generated for each.

```
Chunked Data = (M, RS_M).

Chunked Data Length = len(Chunked Data).

If Message length > RS_M:

Chunked Data = (M, RS_M).

codeword = rs4a.encode(Message).

remainder = codeword[-6:].

remainderhex = remainder.encode().hex().
```

*4) Data encryption:* The Encryption module provides obfuscation and protection of message/data. The framework used the ECIES to accomplish this. In an unsecured channel, ECIES will safeguard the contents from being read or altered with by unauthorized parties. During implementation, the ECIES combines the Secp256k1 curve with the Advanced Encryption Standard – 256 – Galois Counter Mode (AES-256-GCM) to offer the needed security. The AES-256-GCM algorithm is a symmetric encryption algorithm with a 256-bit key size [25]. To achieve ECIES encryption the following steps are followed:

*a) Generation of Key Pairs:* The first step in ECIES is to generate private (PRk) and public key (PUk) pairs. In the elliptic curve, private keys (PRk) that are generated are in integers and public keys (PUk) are points on the curve. Therefore, to encrypt file contents, the Harricent_RSECC model generated both public (PUk) and private keys (PRk). A private key (PRk) was created by obtaining 32 bytes from randomly generated numbers (R). A new private key (PRk) is generated if there is none; otherwise the private key already generated will be loaded. A private key (PRk) is just a sequence of raw bytes. To ensure better security, small numbers must not be chosen as private keys.

```
PRk = rand()

If PRk = NONE then

PRk = rand()

Else

PRk
```

Example: the limit of the integer number (N) as agreed by the communicating parties.

N =
1001579208923731619542312570985008556879071852813756427907490438260516314

PRk = random.randint(1, N)

PRk =
9519399135303906101537810023521385719672175494981093756435606366941355 79

The public key, on the other hand, is determined by multiplying a point on the elliptic curve (G) and the private key (PRk).

PUk = PRk × G

Therefore,

PUk = (PRk × Gx) + (PRk × Gy)

*b) Generation of Shared Secret Keys:* The second step of using the ECIES was to create a shared secret key by multiplying the private (PRk) and public (PUk) key using Elliptic Curve Diffie-Hellman (ECDH) algorithm. The ECDH algorithm is principally used to prevent eavesdropping by facilitating the exchange of the AES shared secret key. This shared key is a public key derived after multiplying the private key of the sender to the public key of the receiver and vice versa. Thus,

Shared Secret key (S) = PRk × PU

*c) Key Derivation Function (KDF) - AES Key and HMAC Key:* The third step of the ECIES algorithm utilized the shared secret key to derive an Advanced Encryption Standard (AES) key and a Hash-Based Message Authentication Code (HMAC) key via a Key Derivation Function (KDF). The KDF component provides another layer of security to prevent man-in-the-middle attack which is a major issue in ECDH. The KDF ensures the hashing of the shared secret key (S) generated using SHA256 algorithm. The steps involved in implementing the key derivation function (KDF) are:

i) Convert Shared key (S) to bytes

ii) Using the SHA256 hash library, the bytes are hashed to produce a hash key (H).

The AES key derivation is based on the key length and the hash (shared) key (H). AES-256-GCM has the key length of 256 bits. To generate an AES key that was used for encryption, the hash (shared) key (H) length should be equal to 256 key bits length.

AES Key Length = 256 bits

AES Key = H [: AES Key Length]

// AES key picks the first part of shared //key whiles the latter is for the HMAC key

More so, HMAC secret key utilized was also based on the key length and the hash (shared) key (H). HMAC's cryptographic strength depends on the size of its output as well as the size and quality of the key. HMAC-SHA256 operates on a bit size of 256. This size produces a sizable output and key. To generate HMAC secret key:

HMAC Key Length = 256 bits

HMAC Secret Key = H [HMAC Key Length:] // HMAC key picks the later //part of the shared key

*d) Message Encryption:* The fourth step in adopting the ECIES is message encryption. To encrypt a message, a random initialization vector (IV) was created and XOR'd with the message in addition to the AES key to generate a ciphertext. Initialization vector (IV) is an abbreviation of "nonce" which connotes the number used once. The randomly generated bits were based on the AES block size and the number of bits in the key length.

The generated IV is converted to bytes depending on the AES block size.

IV = random.getrandbits (AES block size * number of bits per byte).

IV = IV. to_bytes (AES block size).

Ciphertext (C) = (IV XOR message, AES Key).

*e) Hash-Based Message Authentication Code (HMAC):* In the fifth step, HMAC was created to be used for initialization vector (IV) and Ciphertext (C). To create HMAC, two keys are generated from the HMAC Secret Key. The two keys are named, inner key (Ik) and Outer key (Ok). The first 256 hash (Fh) value is generated from the message and Ik. A second 256 hash value is generated from Fh and Ok. These keys are sent to the recipient for verification/decryption.

*f) Transmitting PUk, IV, C and HMAC:* The final step involves the sending of PUk, Initialization Vector (IV), Ciphertext (C) and Hash-Based Message Authentication Code (HMAC). The transmission is through a communication channel to the server facility for storage.

Encrypted data or ciphertext sent ($C_s$) is decrypted using the private key $PR_{k,}$

$Message_{decrypted} = (C_s, PR_{k,})$

*5) Data signature:* This component uses Fast ECDSA to digitally sign files to uphold the integrity of the message. In comparison to other types of digital signatures, the Fast ECDSA technique is designed to conduct fast elliptic curve cryptography. In comparison to ECDSA, it takes very little time to perform its instructions. In the elliptic curve, private keys are integers, and public keys are points on the curve. To offer 128 bits of security, the curve adopted is P256/secp256r1. Creation curves are done using Weierstrass form: $y^2 = x^3 + ax + b (\mathrm{mod}\ p)$.

The Fast ECDSA algorithm employed to digitally sign messages combines P256 / secp256r1 curve and SHA3_256. The curve is imported for use in this study as shown in the example below:

Example:

curve = ec.SECP256R1()

The following are the steps involved in the implementation of Fast ECDSA algorithm:

*a) Generation of Key Pairs:* Private (EPRk) and Public (EPUk) keys were generated to sign and verify messages by using P256 curve. Private (EPRk) keys are integers that were randomly generated and Public (EPUk) key were derived by multiplying the EPRk and a point on the curve (GE). Public

Keys are considered points on the curve.

$EPR_k$ = rand().

$EPU_k = PR_k \times G$.

*b) Signing the Message:* The private key (EPRk) is used to sign the message and the hash function SHA3_256. ECDSA signature (r, s) is generated after signing and returned as 256 bits.

Message = b"This message will be signed and verified".

signature_algorithm = ec.ECDSA(hashes.SHA3_256()).

signature = privatekey.sign(Message, signature_algorithm).

*c) Verifying the Signatures:* To verify that the message has not been compromised, the ECDSA signature (r, s) together with public key ($EPU_k$), decrypted message and hash function SHA3_256 are utilized.

valid = publickey.verify(signature, message, signature_algorithm)

if valid is True then

print("Valid Message")

else

print("Invalid Message")

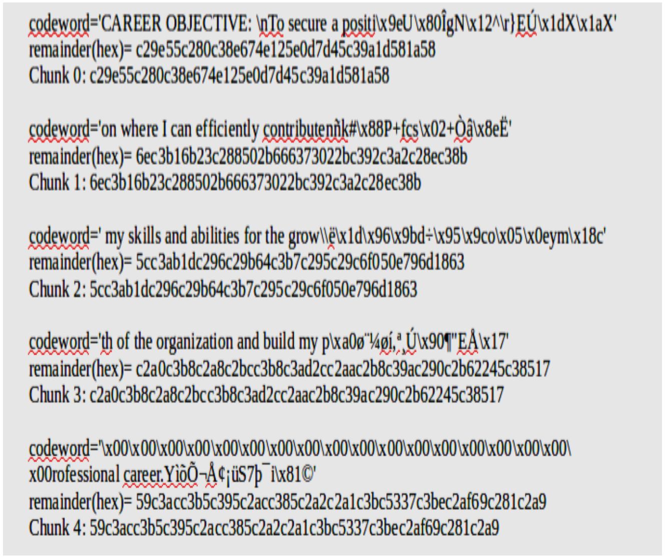
## D. *Securing Data with the Harricent_RSECC Data Protection Model*

To demonstrate the feasibility and performance of Harricent_RSECC Data Protection Model, a python implementation has been developed. Consider the message:

"CAREER OBJECTIVE: To secure a position where I can efficiently contribute my skills and abilities for the growth of the organization and build my professional career"

The message is firstly identified as text through the message identification module. Following the model utilized a "unireedsolomon" library to facilitate in message encoding to generate the codeword and remainders for each message. Several predefined RS coder levels were setup in the Harricent_RSECC Data model. However, in Listing 1 (see Appendix II), an implementation instance RS(15, 9) and RS(255, 251) are presented. The two encoder levels have codeword lengths of 15 and 255 and message lengths of 9 and 251, respectively. The remainders are derived by subtracting the length of the message from the codeword length.

Reference to the case, a user can encode this message and by selecting a coder level of RS(53, 37). The Generator Module is invoked to chunk the message into five different parts for encoding shown in Fig. 6 below. Each of these parts of the message is encoded to the codeword length of 53. This ensures that enough redundant bits are appended to the message to offer the opportunity for data reconstruction in the event of data corruption.



```
codeword='CAREER OBJECTIVE: \nTo secure a positi\x9eU\x80ÎgN\x12^\r}EÚ\x1dX\x1aX'
remainder(hex)= c29e55c280c38e674e125e0d7d45c39a1d581a58
Chunk 0: c29e55c280c38e674e125e0d7d45c39a1d581a58

codeword='on where I can efficiently contributenñk#\x88P+fcs\x02+Òâ\x8eË'
remainder(hex)= 6ec3b16b23c288502b666373022bc392c3a2c28ec38b
Chunk 1: 6ec3b16b23c288502b666373022bc392c3a2c28ec38b

codeword=' my skills and abilities for the grow\\ë\x1d\x96\x9bd÷\x95\x9co\x05\x0eym\x18c'
remainder(hex)= 5cc3ab1dc296c29b64c3b7c295c29c6f050e796d1863
Chunk 2: 5cc3ab1dc296c29b64c3b7c295c29c6f050e796d1863

codeword='th of the organization and build my p\xa0ø¨¼øí,ª¸Ú\x90¶"EÅ\x17'
remainder(hex)= c2a0c3b8c2a8c2bcc3b8c3ad2cc2aac2b8c39ac290c2b62245c38517
Chunk 3: c2a0c3b8c2a8c2bcc3b8c3ad2cc2aac2b8c39ac290c2b62245c38517

codeword='\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\
x00rofessional career.YìõÕ¬Å¢¡üS7þ¯ì\x81©'
remainder(hex)= 59c3acc3b5c395c2acc385c2a2c2a1c3bc5337c3bec2af69c281c2a9
Chunk 4: 59c3acc3b5c395c2acc385c2a2c2a1c3bc5337c3bec2af69c281c2a9
```

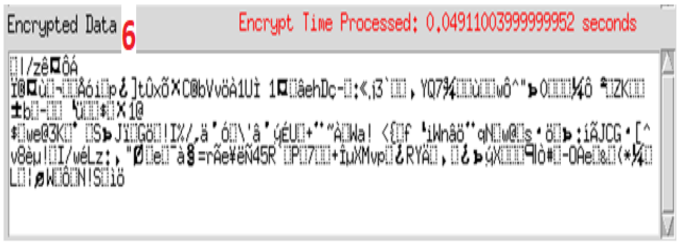
Fig. 6. Codeword and Remainders Generated from the given Message.

Concurrently, the model takes the message, and by utilizing ECIES and Fast ECDSA, the message is encrypted and signed respectively. Fig. 7 shows the output of the encrypted text of the above message.

To encrypt, the model uses the key generating and encrypting libraries from ecies.utils to generate random values, private, public keys and encrypt the message. Listing 2 (see appendix) is a code snippet of the implementation of the data encryption module using ECIES algorithm.

The Fast ECDSA technique used to digitally sign messages combined P256 / secp256r1 curve and SHA3_256. Secret and shared keys are generated automatically from the P256 / secp256r1 curve to sign and verify messages. The message content is signed using the private key and the hash function SHA3_256. Fast ECDSA signature (r, s) is generated after signing and returned as 256 bits. Listing 3 (see Appendix II) is a code snippet of the implementation of signing the message using Fast ECDSA algorithm.

Appendix I provides a python simulation of the implementation of Harricent_RSECC data protection model and the various stages of interactions of the model. Altogether, the remainder, encrypted and signed texts are transmitted for storage. These files provide adequate information to ensure confidentiality and integrity of organizational data.



```
Encrypt Time Processed: 0.04911003999999952 seconds

□|/zê□ôÁ
Ï®□ù□¬□□Åói□p¿]tûxõ×C®bVvöÀ1Uì 1□□âehDç-□:«,¡3`□□,YQ7¾□□ù□□wô^"þO□□□½ô ª□ZK□□
±b□-□□ Ù□□$□×1@
$□we@3K□' □Sþ J'i□Gö□!I%/,ä'ó□\'â'ýéU□+""À□Wa! <{□f 'iWnâö"qN□w@□s•ö□þ:iÃJCG•[^
v8èµ!□I/wéLz:,"Ø□e□¯à§=rÃe¥ëÑ45R □P□7□□+ÎµXMvp□¿RYÄ□,□¿þýX□□□¶ò#□-OAe□&□(*¼□
L□¦øW□ô□N!S□iö
```

Fig. 7. Encrypted Text.

## VI. Discussion

The Harricent_RSECC data protection model provides data owners with the guarantee of data security in transit and storage while preserving data validity. It contributes to the field of security by incorporating encryption, encoding, and signature techniques to provide confidentiality, authenticity, integrity, and non-repudiation levels of security. Should an adversary intercept any of the ciphertexts, the model ensures data secrecy by utilizing ECIES to prevent unauthorized access and eavesdropping. The ECC key size for the Harricent_RSECC data protection model was 256. This provides a 128-bit security equivalent (RSA/DSA will use a key length of 3072). Furthermore, the model employs Fast ECDSA to digitally sign a message, ensuring its integrity and non-repudiation. Fast ECDSA generates hash values from the message which are used for signing the messages and verifying the authenticity of the ciphertexts. The model additionally leverages RS codes to identify and corrects errors that arise in order to provide the capability of retrieving corrupted data files that occurred during transmission or storage, which further enhances data protection in transit or at storage.

The model affirms the studies conducted by [19], [20], [25], [26], [27], [21] that applied elliptic cryptographic algorithms to encrypt and decrypt the selected data and this guarantee the safety of private information, sensitive data, and can enhance the security of communication between computer systems. The Harricent_RSECC model through the use of Fast ECDSA validates the identity of the sender that transmitted the message which upholds the studies conducted by [19], [20] [28]. By employing Fast ECDSA, the content of the message cannot be altered without detection. This ensures the integrity of the message is maintained and, moreover, the signer cannot deny association with the signed content. Moreover, studies conducted by [13], [12], [10], [23] through the utilization of RS codes endorses the benefits of error detection and correction that may emerge.

A comparative discussion supported with existing literature is presented in this section. The security of the Harricent_RSECC data protection model was evaluated using five metrics: Message Identification, faster processing through chunking, obfuscation, detection and correction of compromised messages and signature of message. This is to ensure that the confidentiality, authenticity, integrity and non-repudiation (CAIN) levels of security are provided to the data. By obfuscating message, the confidentiality and authenticity of information stored on computer systems or transferred between its users across the internet is assured. By encoding and decoding the message, the integrity of the message is maintained through the addition of redundant data. By signing the message, the integrity and non-repudiation of the message is ensured. The selected metrics, as adduced in chapter 3, aids in appraising the security potentials of the system.

In analyzing the studies of [19] and [20], it was observed that their system partly used the message identification metric by concentrating on either text or image on separate studies. To ensure fast computational process, their studies utilized message groupings to chunk messages. However, their adopted message groupings will slow the processing of messages as compared to the chunking component (generator module) in the Harricent_RSECC model. In our study, the GM chunks each message based on the RS coder level which leads to faster encoding processing. Even though, the authors utilized the ECC to obfuscate the messages, the two distinct studies lack the functionality of detecting and correcting of compromised messages in an event of error occurrence. More so, [19] and [20] studies failed to offer and maintain integrity of messages by not signing and associating messages transmitted or stored to the sender (i.e., ensuring non-repudiation); which have been ensured through fast ECDSA in Harricent_RSECC model.

In a related study, [23] developed a technology that allows messages to be hidden based on numerical ruler-bundle. It's worthy to note that message hiding is one of the reliable methods of preserving and ensuring the CAIN. Notably, [23]'s study partially made use of only one part of message identification (i.e., image) in comparison with Harricent_RSECC model. Analysis of [23]'s technology also reveals that the authors failed to decompose the information extracted from the image as their system read each stream of input sequence to determine its encoding alphabet; however, their technology hampers efficient computational processes. Their studies enable the hiding of an image inside another image in addition to noise tolerant codes. The model of noise-tolerant codes provided an opportunity to correct up to 25% of errors in the code word. Though, the technology has the capacity to detect and correct errors in comparison with Harricent_RSECC Model, but it lacks the capacity to ensure non-repudiation as their technology does not sign messages that are transmitted and stored.

Also, a study conducted by [21] concentrated on text messages the author's message identification at the exclusion of images. Besides, [21]'s study failed to highlight the imports of chunking which Harricent_RSECC model dwells because chunking is beneficial to ensuring faster computational processes. Like the Harricent_RSECC model, [21]'s model obfuscates the content of the message through the utilization of ECC. However, it lacks the ability to sign, detect and correct compromised messages; hence integrity and non-repudiation levels of security are not maintained.

More so, [22] designed a secure model for data protection in the cloud but the system lacks the ability to differentiate the types of data the study seeks to secure. Data identification is consequential in ensuring CAIN because each data has its own way of processing to offer the needed protection. Also, [22]'s model was not clear on how message chunking is being handled; however, much concentration was given to usability in the study. Whiles the latter is good, the former is equally important to ensure fast computation. More so, [22]'s model obfuscated messages, but the methods employed are AES and RSA, which in literature this study has adduced that the benefits ECC far outweighs RSA. Again, the [22]'s study indicated the concepts of non-repudiation and integrity but the methods in achieving those security metrics are not explicit. Lastly, their system lacks the ability to detect and correct compromised messages which inadvertently affect the integrity of the transmitted or stored data.

TABLE I. COMPARATIVE ANALYSIS OF SOME EXISTING WORK WITH HARRICENT_RSECC MODEL

| **Authors** | **MI** | | **C** | **O** | **EDC** | **S** |
|---|---|---|---|---|---|---|
| [19] | | Image | Yes | Yes – ECC | No | No |
| [20] | Text | | Yes | Yes - ECC | No | No |
| [21] | Text | | No | Yes, ECC | Not | No |
| [22] | No message identification | | Not clear | Yes, AES and RSA | No | Non repudiation and integrity are mentioned but method not clear |
| [23] | | Image | No | Yes - numeric ruler bundle | Yes - numeric ruler bundle | No |
| **Harricent_RSECC** | **Texts and Images** | | **Yes** | **Yes - ECIES** | **Yes - RS** | **Yes – Fast ECDSA** |

Table I presents a comparative analysis of some existing work with Harricent_RSECC model in achieving secured system by basing on the stated metrics:

MI – Message Identification,

C – Chunking,

O – Obfuscation,

EDC – Error Detection and Correction,

S – Signature,

From the levels of security espoused, it can be asserted that it is only the Harricent_RSECC model that combines the functionalities of encrypting, encoding and signing to achieve confidentiality, authenticity, integrity, non-repudiation. Therefore, the Harricent_RSECC data protection framework offers the data owner the assurances of data protection in transit and storage, while maintaining the validity, integrity and confidentiality of the data. It is valid, therefore, to assert that Harricent_RSECC model is the efficient framework comparatively and offers protection to text and image datasets by using both ECC and RS encoding protocols.

To advance this section, the questions posited in section 1.3 are discussed.

## A. *Can RS and ECC be used to Ensure Secure Data Transmission and Secure Data Storage?*

The characteristics of RS and ECC are advanced in Sections 2 and 3. To offer protection to data, several data encoding and encryption techniques have been proposed. In the case of the encryption, such schemes are utilized to obfuscate messages to prevent unauthorized access and modification to protect the validity and secrecy of messages transmitted/stored. While the encoding scheme adds extra bits to the original data to aid in error detection and correction in order to keep the messages from corruption or damage.

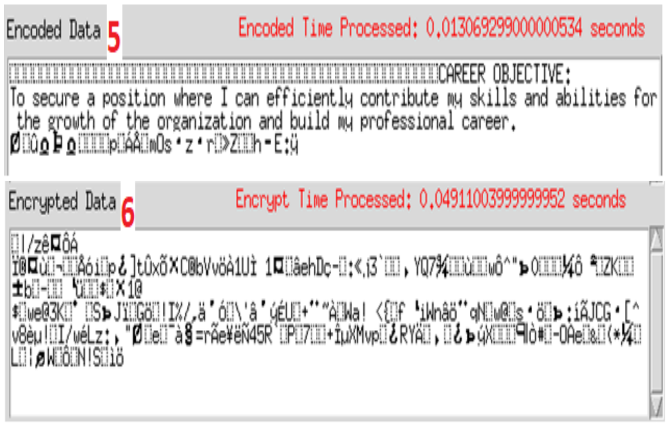
This study evaluates the RS encoding scheme and ECC encryption technique to create a data protection model. Data in transmission and/or storage are susceptible to attacks or security threats. Most security systems focus on either extra bit of data to messages or obfuscate the message. The utilization of one scheme over the other offers security but it is ineffective to provide optimal protection due to the inadvertent loss of data as a consequence of adversary's activities or hardware failures. Therefore, it is essential to combine both encoding and encryption strategies for ensuring effective data protection.

As a public key cryptography, ECC characteristically consists of a public key and a private key which augment the communication between the parties involved. As a result, ECC use significantly smaller parameters than RSA/DSA to achieve the same level of security.

## B. *Can a Proposed Harricent_RSECC Data Protection Model Enhance Data Security by Ensuring an Uncompromised Data Transmission and Data Storage?*

The Harricent_RSECC model and implementation (shown in Fig. 8) achieved the study's purpose of providing enhanced data confidentiality and integrity model by using Reed-Solomon encoding scheme and elliptic curve cryptography. This ensures that the data owner participates in the provision of security to the datasets before storing or outsourcing it to a storage infrastructure [21].

Firstly, from the implementation of the model, a message must be encrypted Using ECIES to inhibit unauthorized access and intrusions if any of the ciphertexts are intercepted. Secondly, prior to transmission, the message is signed by computing a hash value to enable the receiver to authenticate the validity and genuineness of the source message. This aids in detecting any compromises that ensued during transmission by untrusted channel or by an attacker.



Encoded Time Processed: 0.013069299000000534 seconds

```
CAREER OBJECTIVE:
To secure a position where I can efficiently contribute my skills and abilities for
 the growth of the organization and build my professional career.
ØûpÁÅmOs•z•r›Zh-E:ÿ
```

Encrypt Time Processed: 0.049110039999999952 seconds

```
|/zêÔÁ
Ï®ù-Åóip¿]tûxõ×C@bVvöÀ1Uì 1âehDç-:«¡3`,YQ7¾ùwô^"þ0¼ô ZK
±b-ù$X1@
$we@3K SþJïGö!I%/,ä'ó\'â'ýéU+""ÀWa! <{f iWnäö"qNw@s•öþ:íÃJCG•[^
v8èµ!I/wéLz:,"Øe¯à§=rÃe¥ëÑ45R P7+ÎµXMvp¿RYÄ,¿þýX¶ò#-OÀe&(*¼
L¦øWôN!Sïö
```

Fig. 8. A Snapshot of an Encoded and Encrypted Data.

Moreover, in the event of data compromise, the Reed Solomon scheme is utilized to decode and correct the corrupted message. Thus, at the receiving end, the receiver validates the validity and accuracy of the message by computing the hash value on the decrypted message to verify the authenticity of the message. The unmatched values computed from the verification algorithm would render the invalidity of the message.

### *C. Will the Implementation of the Hybrid Data Protection Scheme Offer Necessary Security to Text and Image Datasets?*

The Harricent_RSECC data protection model is a robust model that offers higher security levels to texts and image datasets as it combines the strengths and efficiency of two outstanding data protection schemes. The implementation of the Harricent_RSECC data protection model offers necessary security to both image and text dataset including confidentiality, integrity and non-repudiation. From Appendix I, the Harricent_RSECC model provides confidentiality of the owner's data at the first level of security through encryption by using the ECIES. Again, at the second level of security, the ciphertexts are signed using the fast ECDSA to sign, authenticate and validate the source of the transmitted data. The Harricent_RSECC model provides protection from data loss or corruption using the Reed Solomon coding. Data integrity provided by both fast ECDSA and RS is achieved at the second and third level of the Harricent_RSECC model.

## VII. CONCLUSION

In this paper, we appraise that data is vulnerable to attacks and security threats while in transit and/or storage. The study focused on integrating encoding and encryption schemes to offer optimal protection due to unintended data loss as a result of adversary actions or hardware failures. The study implemented a data security model, dubbed Harricent_RSECC data protection model, to improve CAIN of data by dwelling on five key metrics. Through data identification and classification process, the study was scoped with texts and images. The use of RS codes enabled the detection and correction of compromised messages so as tom maintain the integrity of the message. To prevent man in the middle attack and message eavesdropping, the model obfuscated the message prior to transmission and storage. Consequently, the confidentiality and authenticity of the message guaranteed. Moreover, through message signing, the model achieved the security levels of integrity and non-repudiation.

## VIII. FUTURE WORK

The future study will focus on improving the computational processes so that instead of three deliverables, authors may compress all output into one file. More so, study will expand the scope to cover other scope such audios and videos files.

## REFERENCE

[1] M. M. Kirman, S. M. Saif and A. T. Siddiqui, "Big data : a study of Its issues and challenges," International Journal of Modern Computer Science & Engineering (IJMCSE), vol. 5, no. 1, pp. 29-35, 2016.

[2] V. Amankona, F. Twum and J. B. Hayfron-Acquah, "A framework for securing data by using elliptic curve cryptography and Reed Solomon coding schemes," in 2021 International Conference on Electrical, Computer and Energy Technologies (ICECET), Cape Town, 2021.

[3] R. Denis and P. Madhubala, "Hybrid data encryption model integrating multi-objective adaptive genetic algorithm for secure medical data communication over cloud-based healthcare systems," Multimed Tools Applications, p. 21165–21202, 2021.

[4] D. Sadhukhan, S. Ray, G. Biswas, M. Khan and M. Dasgupta, "A lightweight remote user authentication scheme for IoT communication using elliptic curve cryptography," Journal of Supercomputing, 2020.

[5] C. A. Lara-Nino, A. Diaz-Perez and M. Morales-Sandoval, "Lightweight elliptic curve cryptography accelerator for internet of things applications," Elsevier: Ad Hoc Networks 103, 2020.

[6] C. Nakov, "Elliptic curve cryptography (ECC) - practical cryptography for developers," 2019. [Online]. Available: https://cryptobook.nakov.com/asymmetric-key-ciphers/elliptic-curve-cryptography-ecc. [Accessed 11 12 2021].

[7] J. Fadyn, "Basic elliptic curve cryptography using the TI-89 and maple," 26th International Conference on Technology in Collegiate Mathematics., 2015.

[8] J. Gruska, "Elliptic curves cryptography and factorization," IV054, 2020.

[9] D. Sadhukhan and S. Ray, "Cryptanalysis of an elliptic curve cryptography based lightweight authentication scheme for smart grid communication," in 2018 4th International Conference on Recent Advances in Information Technology (RAIT), 2018.

[10] J. Mounika, "Analysis of modified Reed Solomon error correcting codes," International Journal of Recent Scientific Research, vol. 9, no. 6(A), pp. 27225-27228, 2018.

[11] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields.," Journal of the Society for Industrial and Applied Mathematics, vol. 8, no. 2, p. 300–304, 1960.

[12] W. N and J.-Y. C., "Performance Analysis of (255, 239). Reed Solomon Code for Efficient Knowledge-based Systems in Ubiquitous Environment," International Journal of Innovative Technology and Exploring Engineering (IJITEE), vol. 8, no. 852, 2019.

[13] S. Liu and I. Tjuawinata, "On 2-dimensional insertion-deletion Reed-Solomon codes with optimal asymptotic error-correcting capability," Elsevier: Finite Fields and Their Applications, vol. 73, 2021.

[14] F. Twum, J. Hayfron-Acquah, W. Oblitey and M.-D. William, "Reed Solomon encoding: simplified explanation for programmers.," International Journal of Computer Science and Information Security (IJCSIS), vol. 14, no. 12, 2016.

[15] F. Twum, B. J. Hayfron-Acquah, W. W. Oblitey and R. K. Boadi, "A Proposed Algorithm for Generating the Reed-Solomon Encoding Polynomial Coefficients over GF(256) for RS[255,223]8,32,," International Journal of Computer Applications (0975 – 8887), vol. 156, no. 1, 2016.

[16] W. J. Leis, "Data Transmission and Integrity.," 2018.

[17] D. He, H. Wang, M. Khan and L. Wang, "Lightweight anonymous key distribution scheme for smart grid using elliptic curve cryptography," IET Communications, vol. 10, no. 14, pp. 1795-1802, 2016.

[18] E. S, B. Hureib and A. A. Gutub, "Enhancing Medical Data Security via Combining Elliptic Curve Cryptography and Image Steganography," International Journal of Computer Science and Network Security, vol. 20, no. 8, 2020.

[19] D. S. Laiphrakpam and M. S. Khumanthem, "Image Encryption using Elliptic Curve Cryptography.," Procedia Computer Science, Elsevier, no. 54, p. 472 – 481, 2015b.

[20] D. S. Laiphrakpam and M. S. Khumanthem, "Implementation of Text Encryption using Elliptic Curve Cryptography.," in Eleventh International Multi-Conference on Information Processing-2015 (IMCIP-2015)., 2015a.

[21] R. Obaidur, "Data and Information Security in the Modern World by using Elliptic Curve Cryptography.," Computer Science and Engineering, vol. 7, no. 2, 2017.

[22] A. M. Sauber, P. M. El-Kafrawy, A. F. Shawish, M. A. Amin and I. M. Hagag, "A New Secure Model for Data Protection over Cloud

Computing," Hindawi. Computational Intelligence and Neuroscience., vol. 2021, p. 11, 2021.

[23] O. Riznyk, Y. Kynash, O. Povshuk and Y. Noga, "The Method of Encoding Information in the Images Using Numerical Line Bundles," in IEEE CSIT , Lviv, 2018.

[24] W. Zhang, S. Zou and Y. Liu, "Iterative Soft Decoding of Reed-Solomon Codes Based on Deep Learning," IEEE Communications Letters, 2020.

[25] A. A. Salim and A. B. M. Amal, "Data security for cloud computing based on elliptic curve Integrated encryption scheme (ECIES) and modified identity based cryptography (MIBC).," International Journal of Applied Information Systems (IJAIS), 2016.

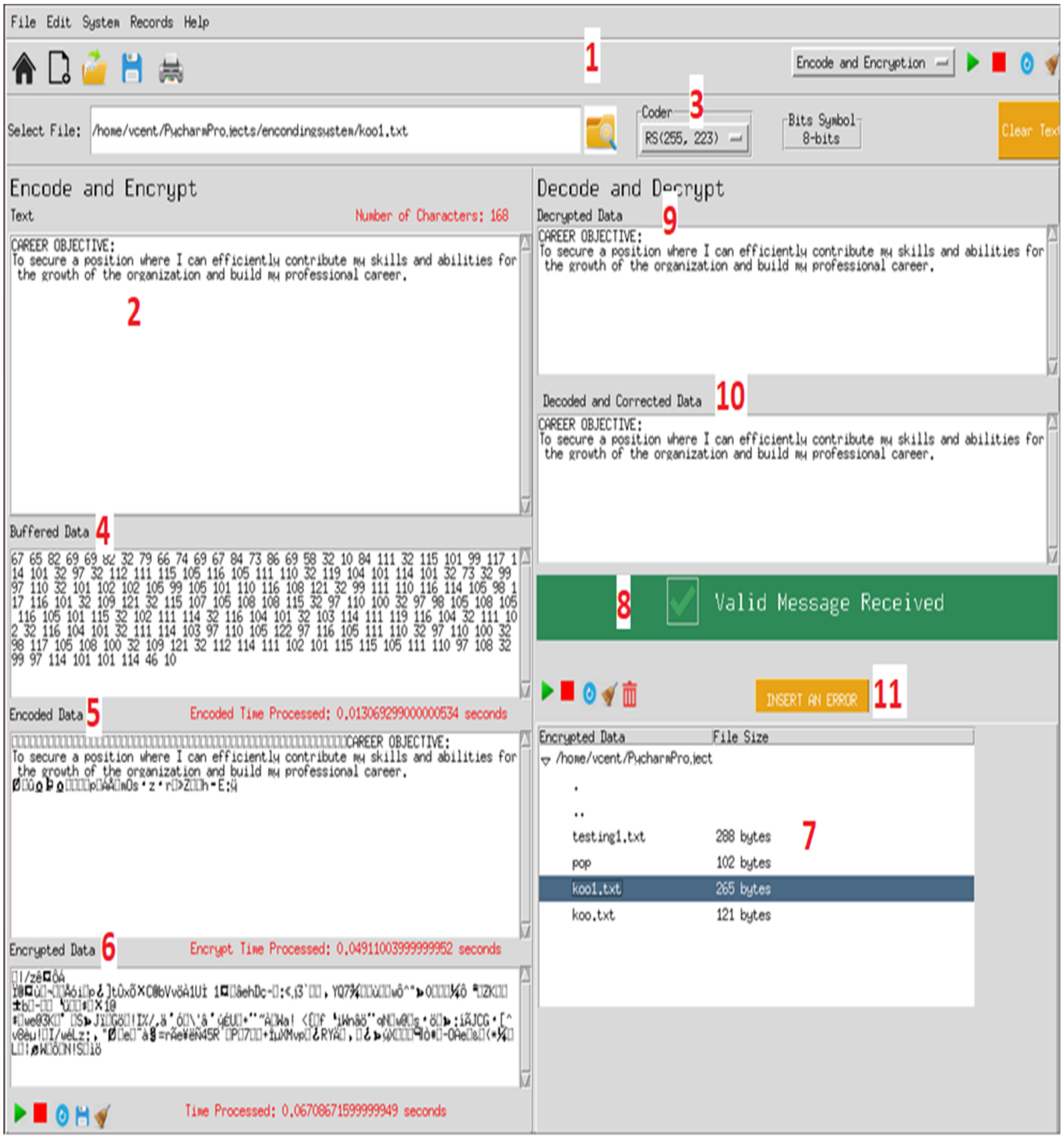
[26] A. G. Amar, C. Noureddine and F. Mezrag, "Performance Evaluation and Analysis of Encryption Schemes for Wireless Sensor Networks," IEEE, 2019.

[27] M. Dindayal, A. K. Danish and K. Y. Dilip, "Security Analysis of Elliptic Curve Cryptography and RSA," Proceedings of the World Congress on Engineering, vol. 1, 2016.

[28] R. S. Soram, K. K. Ajoy and R. S. Soram, "Performance Evaluation of RSA and Elliptic Curve Cryptography.," in 2nd International Conference on Con temporary Computing and Informatics , 2016.

## APPENDIX I

A python simulation of the implementation of Harricent_RSECC data protection model:



Select File: /home/vcent/PycharmProjects/encondingsystem/koo1.txt

Coder: RS(255, 223)

Bits Symbol: 8-bits

Number of Characters: 168

```
CAREER OBJECTIVE:
To secure a position where I can efficiently contribute my skills and abilities for
 the growth of the organization and build my professional career.
```

Buffered Data

```
67 65 82 69 69 82 32 79 66 74 69 67 84 73 86 69 58 32 10 84 111 32 115 101 99 117 1
14 101 32 97 32 112 111 115 105 116 105 111 110 32 119 104 101 114 101 32 73 32 99
97 110 32 101 102 102 105 99 105 101 110 116 108 121 32 99 111 110 116 114 105 98 1
17 116 101 32 109 121 32 115 107 105 108 108 115 32 97 110 100 32 97 98 105 108 105
 116 105 101 115 32 102 111 114 32 116 104 101 32 103 114 111 119 116 104 32 111 10
2 32 116 104 101 32 111 114 103 97 110 105 122 97 116 105 111 110 32 97 110 100 32
98 117 105 108 100 32 109 121 32 112 114 111 102 101 115 115 105 111 110 97 108 32
99 97 114 101 101 114 46 10
```

Encoded Data — Encoded Time Processed: 0.01306929900000534 seconds

```
[illegible]CAREER OBJECTIVE:
To secure a position where I can efficiently contribute my skills and abilities for
 the growth of the organization and build my professional career.
[illegible]
```

Encrypted Data — Encrypt Time Processed: 0.04911003999999952 seconds

```
[illegible]
```

Time Processed: 0.06708671599999949 seconds

Decrypted Data

```
CAREER OBJECTIVE:
To secure a position where I can efficiently contribute my skills and abilities for
 the growth of the organization and build my professional career.
```

Decoded and Corrected Data

```
CAREER OBJECTIVE:
To secure a position where I can efficiently contribute my skills and abilities for
 the growth of the organization and build my professional career.
```

| Encrypted Data | File Size |
|---|---|
| /home/vcent/PycharmProject | |
| . | |
| .. | |
| testing1.txt | 288 bytes |
| pop | 102 bytes |
| koo1.txt | 265 bytes |
| koo.txt | 121 bytes |

Fig. 9. A Snapshot of the Simulation of the Harricent_RSECC Data Protection Model.

## APPENDIX II

Following are some python code snippets for the implementation of Harricent_RSECC Data Protection Model:

```
import unireedsolomon as r
# encoders
rs4a = rs.RSCoder(15, 9, generator=2, prim=0x13, fcr=0, c_exp=4)
rs8d = rs.RSCoder(255, 251, generator=2, prim=0x11d, fcr=0, c_exp=8)
remainderpass = [ ]
codewordpass = [ ]
#the Encoding
        if self.codevariable.get() == "RS(53, 37)":
          if thewordlength > 9:
            #messagebox.showinfo(title="Encoding", message="Message length cannot be greater than 37")
            newfxd = self.chunkmydata(fxd, 37)
            fxdlength = len(self.chunkmydata(fxd, 37))
            if fxdlength > 10:
              messagebox.showerror(message="File Size is too big. Please Choose Another Encoder.")
            else:
              for elem, data in enumerate(newfxd):
                codeword = rs8a.encode(data)
                remainder = codeword[-16:]
                print('remainder(hex)= %s ' % remainder.encode().hex())
                self.codeworddisplay(codeword)
                remainderpass.append(remainder)
                codewordpass.append(codeword)
remainderpass.append(remainder)
            codewordpass.append(codeword)
          encoder = "RS(15, 9)"
```

Listing. 1. The Generation of RS Codeword and Remainder.

```
from ecies.utils import generate_eth_key, generate_key
from ecies import encrypt, decrypt
#encryption-key generation
secp_k = generate_key() #random number
sk_bytes = secp_k.secret  # generate priv-k using the random key
pk_bytes = secp_k.public_key.format(True)  # generate pub-k using the random key
#encrypting data using ECIES
data = bytearray(fxd, "utf8")
encrypted_data = encrypt(pk_bytes, data) #encrypt data using pub-k
self.functiondisplay(newdata, encrypted_data)
```

Listing. 2. The Implementation of the Data Encryption Module using ECIES.

```
from fastecdsa import curve, ecdsa, keys
from fastecdsa.curve import P256
from fastecdsa.keys import export_key, gen_keypair, import_key
from fastecdsa.encoding.der import DEREncoder
from hashlib import sha3_256
#encryption-key generation
# integrity
        private_key, public_key = gen_keypair(P256)
# sign
r, s = ecdsa.sign(fxd, private_key, hashfunc=sha3_256)
```

Listing. 3. The Implementation of the Message Signature using Fast ECDSA Algorithm.