Signal Processing 94 (2014) 174–182

Contents lists available at SciVerse ScienceDirect

Signal Processing

journal homepage: www.elsevier.com/locate/sigpro

Universal data embedding in encrypted domain ☆

Mustafa S. Abdul Karim *, KokSheik Wong

Faculty of Computer Science & Information Technology, University of Malaya, Kuala Lumpur, Malaysia

ARTICLE INFO

Article history:
Received 14 January 2013
Received in revised form
13 June 2013
Accepted 16 June 2013
Available online 24 June 2013

Keywords:
UrDEED
Universal reversible data embedding
Encrypted domain
Golomb–Rice coding
Universal Parser

ABSTRACT

In this work, a Universal Reversible Data Embedding method applicable to any Encrypted Domain (urDEED) is proposed. urDEED operates completely in the encrypted domain and requires no feature of the signal prior to the encryption process. In particular, urDEED exploits the coding redundancy of the encrypted signal by partitioning it into segments referred to as *imaginary codewords* (IC's). Then, IC's are entropy encoded by using *Golomb–Rice codewords* (GRC's). Finally, each GRC is modified to accommodate two bits from the augmented payload. urDEED is designed to preserve the same file-size as that of the original input (encrypted) signal by embedding the quotient part of the GRC's as side information. Moreover, urDEED is consistently reversible and universally applicable to any digital signal encrypted by any encryption method. Experimental results show that urDEED achieves an average embedding capacity of ~0.169 bit per every bit of the encrypted (host) signal.

1. Introduction

Multimedia contents are increasingly produced and communicated in recent years thanks to the availability of efficient capturing devices at low cost and the existence of ubiquitous network environment. To this end, *encryption* is applied as the main conceptual and technical method to preserve privacy, achieve access control and integrity of multimedia contents. Practically, encrypted data are unintelligible. Hence, it is difficult for a third party to extract features, if any, from an encrypted signal without the legitimate decryption key. However, data management system for encrypted signal by a third party is highly demanded nowadays. For example, data are increasingly stored remotely in on-line storage servers (such as cloud storage), which are managed by a third party and the data are often encrypted to avoid unauthorized viewing as well as to protect privacy. Here, *data embedding* can be adopted for managing encrypted signal because it allows the insertion of *extra information*, such as the particulars of the owner, copyright information and other relevant metadata, directly into an encrypted signal.

Date embedding is based generally on modifying some *features* of the host in order to embed the external information [1–5]. However, the extraction of features, if any available, is difficult in the encrypted domain because the encrypted signal is unintelligible. For that, most of the existing data embedding methods in the encrypted domain require partial access to other domains, which may reveal the pre-encryption features of the signal and hence may possibly lead to security breach. In other words, the existing methods rely significantly on the underlying coding structure of the content, the type of media, or the properties of the domain in which the encryption is achieved. For example in [6], features of an image in the frequency domain (i.e., DCT domain) are exploited to achieve irreversible data embedding in the encrypted domain. The encryption is achieved by manipulating selected DCT coefficients, which is guided by the public-key structure without determining a specific

☆ This work was supported by the University Malaya Research Grant (account number RG050-11ICT) under the purview of ICT & Computational Science Research Cluster, UM Research.

* Corresponding author. Tel.: +60 147112279.
E-mail addresses: mustafa.alwahaib@siswa.um.edu.my, mustafa.malaysia@gmail.com, mus515@yahoo.com (M.S. Abdul Karim), koksheik@um.edu.my (K. Wong).

0165-1684/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.sigpro.2013.06.014

standard or method. On the other hand, Lian et al. proposed
a data embedding method for encrypted H.264/AVC compressed videos [7]. In their method, motion, texture and residual information are partially encrypted by manipulating the motion vectors, followed by modifying the prediction modes in an INTRA frame and flipping sign of the coefficients. Then, data embedding is achieved by utilizing the non-zero coefficients in the frequency domain. Similarly, Cancellaro et al. [8] achieve the same commutative property in images stored by using coefficients of the tree structured Haar transform. In [9–11], an image is encrypted by applying XOR operation on its pixels using a pseudo-randomly generated bit sequence determined by a key. Then, external information is embedded by modifying group of pixels in a pre-defined manner. The reversibility property in these methods is achieved by exploiting the spatial correlation among pixel values.

Strictly speaking, a practical data embedding method in the encrypted domain should operate solely in the encrypted domain, where features of the signal prior to the encryption stage should be kept unknown to the data embedder. However, such requirement is not considered in most of the existing methods because they are designed to exploit selected features of the original signal in its pre-encryption stage to realize data embedding. For that, the existing methods are not operating solely in the encrypted domain, but instead they involve two or more domains. In addition, reversibility in [9–11] is not consistently achieved. Hence, the aforementioned methods are not viable in application where any loss of data is prohibited such as medical, military, forensic, etc. More importantly, they are not interchangeable and hence the proposed algorithms are not applicable when considering any other media (e.g., audio, text) or any other encryption methods such as the standard AES [12], Triple-DES [13] or Blowfish [14].

In this paper, we propose a novel Universal Reversible Data Embedding method in the Encrypted Domain (urDEED). Our method operates solely in the encrypted domain in which the coding redundancy is exploited here by entropy coding the encrypted signal and the resulting codewords are modified for data embedding purposes. Performance of the proposed urDEED is verified empirically. urDEED achieves the following features: (1) complete interchangeability, and hence, it is applicable to any signal encrypted by any encryption scheme; (2) consistent reversibility, in which both the original encrypted signal and the embedded data can be restored losslessly, whereas [9–11] may result in restoration error, and, (3) complete file-size preservation to that of the original encrypted (input) signal despite external data embedded into it. To the best of our knowledge, urDEED is the first data embedding method applicable to any encrypted signal, in which no pre-encryption feature is needed to realize reversible data embedding. In other words, urDEED operates solely in the encrypted domain.

The rest of this paper is organized as follows. In Section 2, the possible applications of data embedding in the encrypted domain are presented. In Section 3, a novel framework of data embedding in the encrypted domain is proposed, followed by the application of this framework using urDEED. This includes the mapping of encrypted signal (i.e., ciphertext) into hypothetically defined imaginary codewords (ICs), followed by entropy coding them by using Golomb Rice codewords (GRCs). These GRCs are processed to be of fixed length and then utilized to embed external data. In Section 4, a complete example on the proposed data embedding method urDEED is detailed. Experimental results shown in Sections 5 and 6 concludes this paper.

2. Applications of data embedding in encrypted domain

Ideally, data embedding in encrypted domain should be a process of manipulating an encrypted signal directly to accommodate external information without decrypting the signal or requiring any knowledge about its features prior to the encryption stage. Here, the data embedder can be a third party who is not authorized to access the original content of the signal (i.e., plaintext) and hence has no access to its features in the pre-encryption stage. For example, in medical imaging, images are encrypted to protect privacy of the patient. However, a third-party who manages these images (e.g., system administrator, clerk, nurse) should be able to embed external relevant metadata directly into the encrypted images. Here, the data embedder should neither need to decrypt the image nor exploit any known feature(s) of the original signal in order to achieve data embedding. As another application, in the cloud computing scenario, users of the facility may encrypt their data to protect privacy. Hence, embedding classification, ownership information, data retrieval information or any other relevant information into the encrypted data by cloud-computing service provider should be achieved practically without the need to decrypt the data. Another possible application is to preserve the integrity of encrypted signal by computing the hash value of the encrypted signal and embedding the hash directly into the encrypted signal. Similarly, cyclic redundancy check (CRC) or other error-correction information can be embedded in the encrypted signal to serve the same purpose.

3. Framework of urDEED

In general, an encrypted signal appears random and it should be unintelligible to any party. As such, it is difficult to parse the bitstream of an encrypted signal into a meaningful form. Based on this observation, we propose a framework in Fig. 1 that hypothetically defines some features in the encrypted signal in order to process it. Specifically, the input encrypted signal is virtually parsed. Here, the virtual parsing process is achieved by modeling the signal into imaginary codewords, which are in turn entropy coded to exploit coding redundancy. In this work, we consider the Golomb–Rice codeword (GRC) for entropy coding purposes. Next, the variable-length GRC's are converted into fixed-length GRC's in order to preserve file-size to that of the original (encrypted) signal. Finally, external information is embedded into the fixed-length GRC's. Detail of each operation is described in the following subsections.

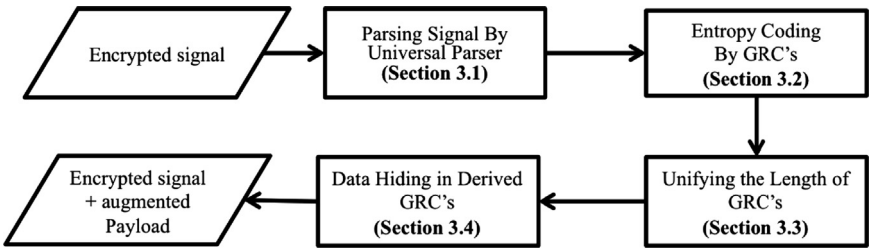

Fig. 1. Flow of operations in urDEED.

3.1. Parsing signal by universal parser

Since the extraction of features from unintelligible encrypted signal is difficult, we assign new feature(s) to the encrypted signal and achieve data embedding by modifying these features. The assignment process is *reversible* and hence the original encrypted signal can be perfectly restored as shown in Section 4.

The assignment of new features is achieved by a parser referred to as *universal parser*. By applying the universal parser, a set of imaginary codewords (IC's) is obtained by partitioning the bitstream of the encrypted signal into fixed length non-overlapping segments (each of length L bits) where each L-bit segment eventually becomes the IC. The total number of IC's is denoted by λ_L and it is computed as follows

$$\lambda_L = \left\lceil \frac{N}{L} \right\rceil, \tag{1}$$

where N is the total number of bits in the encrypted signal, $1 \le L \le N$ and $\lceil Q \rceil$ rounds Q to the nearest integer towards positive infinity. The last IC in the bitstream is called the *end-of-signal* IC, which may not be of length L. The sequence number of end-of-signal IC is recorded as side information and treated as part of the payload. Formally, we define the universal parser as an ordering function as follows:

Definition 1. Suppose A is a set of alphabets and $X = \{x_g\}_{g=1}^{N} \subseteq A$ is a sequence of finite and statistically independent symbols where $N \ge 2$. A universal parser is an ordering function $\mathbb{D}(X, L)$ that partitions X into non-overlapping ordered tuples where each tuple consists of L symbols from X such that

$$\mathbb{D}(X, L) = X_L = \left\{ T_1^L, T_2^L, \ldots, T_{\lambda_L}^L \right\}. \tag{2}$$

Informally, T_ν^L is an imaginary codeword (IC) of length L. Formally, T_ν^L is an ordered tuple that consists of L elements from X and is defined as follows:

$$T_\nu^L = (x_\nu^1, x_\nu^2, \ldots, x_\nu^L). \tag{3}$$

Here, the following equation relating the cardinalities holds true:

$$|\{\{x_1^1\}, \{x_1^2, \ldots, x_1^L\}\}| + |\{\{x_2^1\}, \{x_2^2, \ldots, x_2^L\}\}| + \cdots + |\{\{x_i^1\}, \{x_i^2, \ldots, x_i^L\}\}| = |X|, \tag{4}$$

where $\{\{x_\nu^1\}, \{x_\nu^2, \ldots, x_\nu^L\}\}$ is the Kuratowski's representation of the tuple T_ν^L and $|U|$ is the cardinality of the set U. Here, the following relation also holds true:

$$\bigcap \{T_\nu^L\}_{\nu=1}^{\lambda_L} = \bigcap \{\{x_\nu^1\}, \{x_\nu^2, \ldots, x_\nu^L\}\}_{\nu=1}^{\lambda_L} = \phi, \tag{5}$$

where each element x_ν^d is unique with respect to its position d. □

In the following theorems, we prove that the entropy X_L depends on L.

Theorem 1. *Suppose $X = \{x_1, x_2, \ldots, x_N\}$ is a discrete uniformly distributed set of elements such that the probabilities $p(x_1) = p(x_2) = \cdots = p(x_N) = 1/N$. If a universal parser orders X such that*

$$X_L = \mathbb{D}(X, L) \tag{6}$$

and

$$X_{L+1} = \mathbb{D}(X, L+1), \tag{7}$$

then

$$I(T_i^L) > I(T_j^{L+1}) \tag{8}$$

holds true $\forall T_i^L \in X_L$ and $\forall T_j^{L+1} \in X_{L+1}$ where $I(T_i^L)$ and $I(T_j^{L+1})$ are the self-information of the tuples T_i^L and T_j^{L+1}, respectively.

Proof. Since X is discrete and has uniform distribution, $\forall T_i^L \in X_L$,

$$p(T_i^L) = \frac{1}{\lambda_L}, \tag{9}$$

and $\forall T_j^{L+1} \in X_{L+1}$,

$$p(T_j^{L+1}) = \frac{1}{\lambda_{L+1}} \tag{10}$$

hold true. Increasing L to $L+1$ results in

$$\lambda_L > \lambda_{L+1}. \tag{11}$$

Hence,

$$\frac{1}{\lambda_L} < \frac{1}{\lambda_{L+1}}. \tag{12}$$

By taking the logarithm of the both sides of Inequality (12), the following is derived:

$$\log_2 \left[\frac{1}{\lambda_L} \right] < \log_2 \left[\frac{1}{\lambda_{L+1}} \right] \tag{13}$$

$$\Rightarrow -\log_2 \left[\frac{1}{\lambda_L} \right] > -\log_2 \left[\frac{1}{\lambda_{L+1}} \right] \tag{14}$$

$$\Rightarrow -\log_2(p(T_i^L)) > -\log_2(p(T_j^{L+1})) \tag{15}$$

and hence $I(T_i^L) > I(T_j^{L+1})$ □

Inequality (15) indicates that the amount of self-information associated with the outcome of elements ordered at length L is higher than that of $L+1$. In other words, X_L has higher entropy than X_{L+1} as shown in the next theorem.

Theorem 2. *If a universal parser is utilized to order X into X_L and X_{L+1} as defined in Eqs. (6) and (7), respectively, then*

$$H(X_L) > H(X_{L+1}) \tag{16}$$

where X is a set of discrete and uniformly distributed symbols, $H(X_L)$ and $H(X_{L+1})$ are the entropies of the two set of tuples X_L and X_{L+1}, respectively, for $1 \le L \le N$.

Proof. Initially, $H(X_L)$ is defined as [15,16]

$$H(X_L) = \frac{1}{L}\sum_{i=1}^{\lambda_L} p(T_i^L) \times \theta(T_i^L) \tag{17}$$

where $p(T_i^L)$ is the probability of the tuple T_i^L in X_L and $\theta(T_i^L)$ is the length of the codeword required to encode the tuple T_i^L. Since X is discrete and uniformly distributed, $p(T_i^L)$ and $\theta(T_i^L)$ are constants $\forall T_i^L$. That is,

$$p(T_i^L) = \frac{1}{\lambda_L} \tag{18}$$

and

$$\theta(T_i^L) = \log_2(\lambda_L). \tag{19}$$

Hence, Eq. (17) becomes

$$H(X_L) = \frac{\lambda_L \times \log_2(\lambda_L)}{L \times \lambda_L} \tag{20}$$

$$\Rightarrow H(X_L) = \frac{\log_2(\lambda_L)}{L} \tag{21}$$

Next, we prove Inequality (16) based on Inequality (11):

$$\log_2(\lambda_L) > \log_2(\lambda_{L+1}) \tag{22}$$

$$\Rightarrow \frac{\log_2(\lambda_L)}{L} > \frac{\log_2(\lambda_{L+1})}{L} \tag{23}$$

however

$$\frac{\log_2(\lambda_{L+1})}{L} > \frac{\log_2(\lambda_{L+1})}{L+1}. \tag{24}$$

Hence, the following holds true by transitivity:

$$\frac{\log_2(\lambda_L)}{L} > \frac{\log_2(\lambda_{L+1})}{L+1}. \tag{25}$$

Therefore, Inequality (25) can be rephrased according to Eq. (21) as

$$H(X_L) > H(X_{L+1}) \quad \square \tag{26}$$

Corollary 1. *The following inequality holds true*:

$$H(X_1) > H(X_2) > \cdots > H(X_N) \tag{27}$$

Proof. Inequality (27) follows directly from Inequality (26) □

Corollary 2. *The upper and lower bounds of Inequality (27) are given as follows*:

$$\log_2(\lambda_1) > H(X_2) > \cdots > H(X_{N-1}) > 0 \tag{28}$$

Proof. The upper and lower bounds of Inequality (27) are defined by taking the limits of Eq. (21) to 1 and ∞, respectively, as follows:

$$\lim_{L\to 1} \frac{\log_2(\lambda_1)}{L} = \log_2(\lambda_1) \tag{29}$$

$$\lim_{L\to \infty} \frac{\log_2(\lambda_\infty)}{L} = 0 \quad \square \tag{30}$$

Example 1. Here, we show an example where the entropy $H(X_L)$ changes as L changes. Given an ordered tuple $X = (A, B, C, D)$. X can be processed by the universal parser as follows:

At $L=1$, $X_1 = \mathbb{D}(X, 1)$ has 4 tuples, namely, $T_1^1 =$ "A", $T_2^1 =$ "B", $T_3^1 =$ "C" and $T_4^1 =$ "D". Hence, $\lambda_1 = 4$ and $H(X_1) = \log_2(4)/1 = 2$ bits/tuple.

At $L=2$, X_2 has 2 tuples, namely, $T_1^2 =$ "AB", $T_2^2 =$ "CD". Hence, $\lambda_2 = 2$ and $H(X_2) = \log_2(2)/2 = 0.5$ bits/tuple.

At $L=3$, X_3 has 2 tuples, namely, $T_1^3 =$ "ABC", $T_2^3 =$ "$D\Delta\Delta$" where Δ is an empty symbol that can be ignored. Hence, $\lambda_3 = 2$ and $H(X_3) = \log_2(2)/3 = 0.3$ bits/tuple.

At $L=4$, X_4 has 1 tuple, namely, $T_1^4 =$ "$ABCD$". Hence, $\lambda_4 = 1$ and $H(X_4) = \log_2(1)/4 = 0$ bits/tuple.

Corollary 3. *As a consequence of Inequality (28), the following inequality holds true*:

$$\rho_1 < \rho_2 < \cdots < \rho_N \tag{31}$$

where ρ_L is the amount of redundancy in a set of tuples X_L and it is defined as follows:

$$\rho_L = \Theta_{avg}(X_L) - H(X_L) \tag{32}$$

where $\Theta_{avg}(X_L)$ is the average length of the codewords encoding the tuples of X_L and $H(X_L)$ is defined in Eq. (21) □

Corollary 4. *Since X can be modeled into X_L by Eq. (2) for $1 \le L \le N$, then the following holds true*:

$$\mu \ge N + 1 > 2, \tag{33}$$

where μ is the number of possible models to which X can be transformed.

Proof. By applying the universal parser, X can be transformed into N models by Eq. (2) where in each model, symbols of X are ordered into tuples of unified lengths L. On the other hand, there is at least one parser $\hat{\mathbb{D}}(X, \sigma)$ that differs from the universal parser, i.e., $\hat{\mathbb{D}}(X, \sigma) \neq \mathbb{D}(X, L)$ where σ is a parsing coefficient (say a seed to generate a sequence of random lengths each less than N). By applying $\hat{\mathbb{D}}(X, \sigma)$, the symbols of X are ordered into tuples of non-unified lengths. Hence, by considering the two classes of parser together, i.e., $\hat{\mathbb{D}}(X, \sigma)$ and $\mathbb{D}(X, L)$, there are at least $N+1$ models to which X can be transformed. On the other hand, by Definition (1), $N \ge 2$ and hence $N + 1 > 2$, which proves Inequality (33). □

For example, suppose a JPEG image is represented by a bitstream of length N bits. Using Eq. (2), the image can be modeled into N different ways. On the other hand, one

additional model can be obtained by applying the standard (i.e., format-compliance) parser to the image.

Generally, Inequality (33) indicates that for any set of symbols X of length N bits, X can be transferred into at least $N+1$ different models. When the universal parser is applied, the maximum number of models is limited to N models, where each model is obtained by setting L to a certain value in Eq. (2). Changing the length L leads to a change in the entropy $H(X_L)$ of a set of tuples as discussed earlier. Hence, entropy coding the set of tuples by a non-ideal coding scheme ζ leads to the removal of only certain amount of redundancy ρ_L. In this context, the length L that removes the most redundancy is referred to as the *optimum L*. In urDEED, the optimum L is defined empirically. Here, the removed redundancy makes spaces for data embedding as detailed in subsection 3.2.

3.2. Entropy coding

In urDEED, the set of imaginary codewords (IC's) are entropy coded for three purposes: (A) to represent the IC's in a format that can be modified in order to embed the augmented payload; (B) to preserve the original file-size of the encrypted signal, and; (C) to remove some redundant data in order make room for accommodating the augmented payload. Hence, we emphasize that entropy coding in urDEED does not aim at compressing the encrypted signal. In this work, Golomb–Rice codewords (GRC's) are utilized for entropy coding purposes. The entropy coding process starts with the construction of the histogram of the set of IC's. Then, IC's of higher frequency occurrences in the bitstream are coded by GRC's of shorter length, and vice versa. For reversibility purposes, the sorted IC's (based on their frequency) are stored as side information along with the augmented payload.

By definition, each GRC consists of three pre-defined parts, namely; (1) quotient part (q) which consists of a sequence of zeros of length z; (2) termination bit (b) of constant value 1; and (3) reminder part (r) which consists of an arrangement of bits of constant length n [17]. Hence, any GRC is of the following form:

$$(qbr) = (0_1 0_2 \cdots 0_z 1 r_1 r_2 \cdots r_n). \tag{34}$$

We impose that $z > 0$ because q and b are utilized for data embedding purposes as detailed in subsection 3.4. For example, if $n=2$, then the set of GRC's consists of (0100), (0101), (0110), (0111), (00100) and so forth. Note that the length of GRC's is increased by appending '0' to the quotient part for every 2^n codewords generated.

3.3. Unifying the length of GRC's

As described in the previous section, GRC's are variable-length codewords. Hence, some GRC's are of length $>L$ and they cause file-size increment. In order to preserve the file-size of the original encrypted signal, we impose that all GRC's must be of constant length $L = n+2$ bits. Hence, the quotient part q of each GRC is trimmed and appended to the augmented payload. The trimming process is detailed in the following subsections.

3.3.1. Trimming q's

Trimming is achieved by removing z bits, in the order from left to right, (i.e., the entire quotient part q) of each GRC. The trimmed q is replaced with a dummy value of 0. The trimmed q's may be flipped to mark the end of a codeword and the beginning of another codeword as shown in the next subsection. Thus, after the trimming process, all GRC's are in the general form of $(0br) = (01 r_1 r_2 \cdots r_n)$. For example, if $L=4$ and given GRC = (000100), then such GRC is trimmed to (0100), where the trimmed 000, i.e., the entire quotient part, is appended to the augmented payload.

3.3.2. Appending trimmed q's to augmented payload

Given $GRC_i = (q_i b_i r_i)$ for $i = 1, 2, \ldots, N$, the quotient part of these GRC's are appended to the augmented payload as $q_i \overline{q_{i+1}} q_{i+2} \overline{q_{i+3}} \cdots \overline{q_N}$ (assuming N is an even number), where $\overline{A_i}$ is the logic inverse of A_{i-1}. The logical inverse operation plays a crucial role in signifying the end of the quotient part for a GRC. In particular, the length of zero-run (or one-run) determines the number of zeros (i.e., length z) in the original quotient part (which was earlier trimmed), and the change from '0' to '1' (and vice versa) marks the beginning of the next quotient part. This appending operation is further illustrated in the following examples.

Example 2. Constant length scenario: Given {(**0**100), (**0**101), (**0**110)} as the set of GRC's. The corresponding q's are **0**, **1**, and **0**, respectively. Note that the original q of the second GRC (i.e., 0101) is 0, but it is logically inversed (i.e., flipped) to $q=1$.

Example 3. Variable length scenario; Suppose {(**00000** 1 00), (**00** 1 01), (**000** 1 10), (**000** 1 00)} is a set of GRC's. Then, q of these GRC's are appended to the augmented payload as **00000**, **11**, **000** and **111**.

3.4. Data embedding in derived GRC

The augmented payload now consists of the trimmed quotient parts, side information (i.e., sorted IC's and λ_L), and the actual external information as illustrated in Fig. 2. This augmented payload is then embedded in the set of fixed-length GRC's. Each fixed length GRC = $(0br) = (01 r_1 r_2 \cdots r_n)$ can accommodate two bits from the augmented payload. In particular, the dummy '0' and 'b' bits in the trimmed GRC are replaced by the information (i.e., two bits) to be embedded. In other words, a modified GRC becomes $(p_j p_{j+1} r_1 r_2 \cdots r_n)$ where p_j and p_{j+1} are the two bits obtained from the augmented payload.

The first component of the augmented payload (i.e., the trimmed quotient parts) is embedded, followed by the side information. Here, the length of those trimmed q's is at least λ_L bits (since $x > 0$ is enforced), and the length of the side information is $L \times M$ bits where M is the number of the imaginary codewords that actually occurs

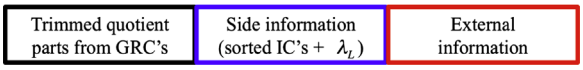

Fig. 2. Layout of the augmented payload.

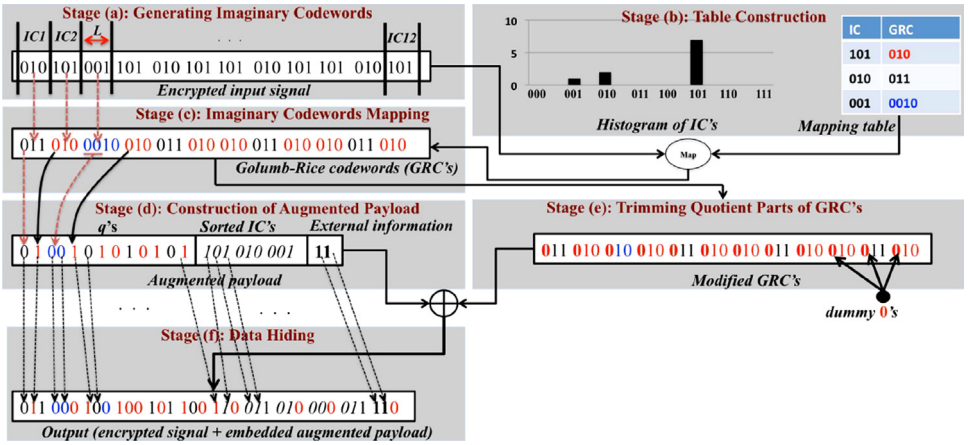

Fig. 3. Example on data embedding in urDEED.

(i.e., frequency of occurrence greater than zero) and $1 \le M \le 2^L$. Finally, the external information is embedded. The length of the embeddable external payload is referred to as the effective embedding capacity C and it is computed as follows:

$$C = (\lambda_L \times 2) - [\kappa + (L \times M)] \text{ bits} \tag{35}$$

where κ is the length of those trimmed quotient parts in the augmented payload and $\kappa \ge \lambda_L$. Note that achieving $C > 0$ implies that the following inequality must hold true:

$$\lambda_L \times 2 > \kappa + (L \times M) \tag{36}$$

Here, Inequality (36) depends on two factors: (a) λ_L, which depends on the chosen length for each imaginary codeword (i.e., L), and; (b) κ, which depends on the distribution of the imaginary codewords. If the distribution of the imaginary codewords is concentrated around a few arrangements (among the total of 2^L arrangements), then κ is small, and vice versa. Hence, choosing L that satisfies Inequality (36) plays the major role in controlling the effective embedding capacity C.

4. Example on data embedding/extracting in urDEED

Fig. 3 shows an example to walkthrough the proposed data embedding method. This example is based on a segment of random bitstream X shown in *Stage* (*a*) but the description is general enough to handle the entire encrypted content (i.e., sequence of bits). Here, the set of imaginary codewords (IC's) is generated by applying the universal parser, which partitions X in segments, each of 3 bits (i.e., $L=3$), and $X_3 = \mathbb{D}(X, 3)$. In this example, $N=36$ and hence $\lambda_3 = 36/3 = 12$ IC's.

In *Stage* (*b*), the IC's are sorted according to their frequency of occurrences, starting with IC of the highest occurrence. Then, the mapping table (from IC's to GRC's) is constructed (*Stage* (*b*)) and this table is stored as part of the augmented payload in the actual implementation (as shown in *Stage d*). In this example, three GRC's are generated, each of length $n=1$ for its remainder part. Since IC = 101 occurs 7 times (i.e., highest occurrence), it is assigned to the first GRC = 010. In similar fashion, IC = 010 occurs 4 times and hence it is assigned to GRC = 011. The longest GRC = 0010 is assigned to IC = 001 because it occurs only once.

In *Stage* (*c*), the set of imaginary codewords are mapped into GRC's as stipulated by the mapping table. In *Stage* (*d*), the various components of the augmented payload are concatenated, which include the set of trimmed q's (generated in *Stage e*), the set of sorted IC's in the table and external information of two bits '11'. Note that the length of GRC = 0010 is 4 bits, which is longer than $L=3$ bits. Hence in *Stage* (*e*), the quotient part of each GRC is trimmed and added to the augmented payload in order to preserve the original size of the input encrypted signal. Thus, $GRC_1 = 011$ and $q_1 = 0$, and $GRC_2 = 010$ and $q_2 = 1$, which is the logical inverse of q_1. For $GRC_3 = 0010$, $q_3 = 00$ are derived, which is the logical inverse of q_2, and so on. The derived q's are concatenated to form the first component of the augmented payload as shown in *Stage* (*d*). Next, a dummy value of '0' is assigned to each modified (or fixed-length) GRC. Finally, in *Stage* (*f*), data embedding is achieved by embedding the augmented payload in the set of fixed-length GRC's derived in *Stage* (*e*). Here, each GRC accommodates two bits of the payload. These two bits are embedded by replacing the dummy '0' and the constant '*b*' in each modified GRC.

Note that the effective embedding capacity in this example is computed as follows: $\lambda_3 = 12$, $\kappa = 13$, $L=3$ and $M=3$. Hence, $C = (12 \times 2) - [13 + (3 \times 3)] = 2$ bits. Here, C is of small amount because there are only a few IC's in the bitstream considered. For simplicity, data embedding is achieved in this example without coding L and M at the beginning of the augmented payload. Empirically, we observed in the experimental results (Section 5) that maximum C is consistently achieved at $L=3$ and hence the length of the side information is at most $L \times M = 3 \times 2^L = 24$ bits for this particular setting.

The extraction of the payload and restoration of the original encrypted signal in urDEED are almost the opposite of the embedding process. We briefly walkthrough the decoding process using the same example:

(1) Encoded bitstream in *Stage* (*f*) (i.e., Output) is stored in the memory for reconstruction purposes.

(1.a) The original GRC's are reconstructed by examining the dummy 0's and *b*'s of the set of GRC's obtained in *Stage* (*f*). A change from 0 to 1 (or vice versa) marks the beginning of the quotient part for the next GRC where the length of zero-run (or one-run) indicates the original number of zeros in the quotient part (i.e., *z*) of the current GRC. This is equivalent to performing the reversed steps of *Stage* (*f*), (*e*+*d*) and (*c*). The process is repeated until the quotient part of λ_L codewords are restored.

(1.b) Next, the first two bits (i.e., originally the dummy '0' and termination bit '*b*') from each of the remaining fixed-length GRC's are examined to extract the side information. After retrieving 24 bits (in the case of $L=3$), the table in *Stage* (*b*) is re-generated by utilizing the extracted side information.

(1.c) The processes are repeated for the remaining GRC's to retrieve the actual external information.

(2) Based on the re-generated table, GRC's are losslessly re-mapped into IC's using the reverse steps from *Stage* (*c*) to (*a*).

5. Experimental results

As a proof of concept, the proposed method is implemented using C/C++ programming language. The performance of urDEED is verified from the following aspects: (1) interchangeability; (2) reversibility; (3) effective embedding capacity; and (4) ability to preserve file-size. Since urDEED is applied to encrypted signal, the distortion of the signal due to data embedding (i.e., by applying urDEED) is irrelevant and hence not measured. However, it is verified that the embedded augmented payload can be perfectly extracted and the original input (encrypted) signal can be losslessly reconstructed. In addition, the reconstructed (encrypted) signal can be decrypted, which verified that the proposed urDEED is completely reversible. As mentioned earlier, there is no prior work operating in the encrypted domain that offers the same features. Hence, we compare the performance of the proposed urDEED to that of Zhang's method [9] because it is the closest match in the literature. The following subsections detail the performances of urDEED.

5.1. Interchangeability

Interchangeability means that urDEED is applicable to any encrypted signal independently from the features of (A) the original media of the signal, and (B) the applied encryption scheme. To verify (A), urDEED is applied to encrypted signals which are originally three different media, namely, image, audio, and text. The image is Tiffany, which is a gray-scale raw image of 512×512 pixels, the audio is of Waveform Audio File Format (wave) of 2 s in length with bit rate of 1411 Kbps, and the texts are two Microsoft WordÁdocuments of size 12.7 and 292 KBytes.

To verify (B), each of the aforementioned media is encrypted by various encryption algorithms. The considered encryption schemes are Blowfish [14], Advanced Encryption Standard (AES) [12], Rivest's cipher (RC) 2,4,6 [18], Data Encryption Standard (DES) [19], Triple-DES [13], Serpent [20], Twofish [21], RSA [22], CAST [23], Information Concealment Engine (ICE) [24], MARS [25], MISTY [26] and Tinny Encryption Algorithm (TEA) [27]. In addition to these methods, hybrid cascade encryption algorithms, such as AES-TwoFish-Serpent, are also applied. The output ciphertext of each encryption scheme is manipulated by urDEED to embed the augmented payload.

Table 1 records the experimental results using urDEED to embed the augmented payload into the encrypted image, audio and texts. Results show that, generally, urDEED is

Table 1
Effective embedding capacity of urDEED when applied to different media encrypted by using various encryption schemes.

Encryption method	Image		Audio		Text	
	C	bpb	*C*	bpb	*C*	bpb
Blowfish-448	335,809	0.168	474,112	0.168	17,703	0.170
AES-128	335,971	0.168	473,672	0.168	17,650	0.170
AES-192	335,216	0.168	474,659	0.168	17,956	0.173
AES-256	336,221	0.168	472,252	0.167	17,878	0.172
RC2-1024	336,190	0.168	473,750	0.168	17,891	0.172
RC4-2048	335,574	0.168	474,963	0.168	17,929	0.172
RC6-2048	336,445	0.168	476,807	0.169	17,864	0.172
DES-56	335,700	0.168	473,870	0.168	17,709	0.170
TDES-256	336,216	0.168	473,393	0.167	17,743	0.171
Serpent-256	400,422	0.167	472,349	0.167	400,316	0.167
Twofish-256	400,621	0.167	473,286	0.167	17,630	0.169
Blowfish-AES-256	401,167	0.168	474,274	0.168	401,370	0.168
Serpent-Flowfish-AES-256	401,137	0.168	473,432	0.168	400,755	0.168
AES-Serpent-256	401,431	0.168	474,554	0.168	400,309	0.167
AES-TwoFish- Serpent-256	400,916	0.168	473,057	0.167	400,971	0.168
Serpent-TwoFish-256	399,502	0.167	473,573	0.168	400,531	0.167
RSA-1024	367,767	0.168	518,858	0.184	19,706	0.189
CAST-256	334,709	0.167	473,877	0.168	17,994	0.173
ICE-64	336,440	0.168	475,146	0.168	17,771	0.171
MARS-1248	336,901	0.169	475,569	0.168	17,977	0.173
MISTY-128	335,999	0.168	473,432	0.168	18,060	0.174
TEA-128	336,019	0.168	474,186	0.168	18,084	0.174

Table 2
Functional comparison between Zhang's method [9] and the proposed urDEED.

Method	Average bpb	Applicable to any media?	Applicable to any encryption scheme?	File-size preserving?	Reversible?
Zhang [9]	0.006[a]	No (image only)	No	Yes	Conditional
urDEED	0.173	Yes	Yes	Yes	Always

[a] This is the highest embedding capacity reported in [9] while sacrificing reversibility. For the case of perfect reversibility, the highest embedding capacity reported is 0.004 bpb.

Table 3
Effective embedding capacity of urDEED for various L in encrypted Lenna image.

L	3	4	5	6	7	8
C	352,452	91,323	3224	−30,866	−44,394	−49,105

interchangeably applicable to a signal encrypted by various encryption methods because the program is able to execute to its completion and the effective embedding capacity of value $C>0$ is achieved. This suggests that the limitation of interchangeability in the existing methods [9–11] is addressed by the proposed urDEED.

For fair comparison purposes, we also embed information into two images (i.e., Lenna and Man of dimensions 512×512 pixels) encrypted by Zhang's XOR operations described in [9] and the results are recorded in the second column of Table 2. In terms of interchangeability, Zhang's method is restricted to one encryption scheme (i.e., XOR-based encryption proposed in [9]) while urDEED is universally applicable to any encryption scheme. Also, Zhang's method [9] is only applicable to image stored in the spatial domain while the proposed method can be applied to any media/signal coded in any domain.

5.2. Reversibility

Reversibility is the ability to perfectly reconstruct the original encrypted signal. The reversibility in urDEED is due to the utilization of the lossless GRC for data embedding, and this reversible functionality is verified by using various media and various encryption algorithms. On the other hand, Zhang's method [9] is not able to guarantee a perfect reconstruction of the original image due to the dependency of correlation among neighboring pixels.

5.3. Effective embedding capacity

The effective embedding capacity C depends on the length L as shown in Eq. (35). To find the optimum L (defined in subsection 3.1), we apply urDEED to Lenna image (512×512 grayscale, raw) encrypted by Blowfish [14] using different L values as the representative case. Table 3 shows the resulting C at each L. It is observed that C at $L=4$ is 91,323 bits, which is lower than its counterpart at $L=3$. C decreases to 3224 bits at $L=5$. It is observed that $C<0$ for $L\in\{6,7,8\}$, which indicates that there is no venue for embedding extra information in the encrypted image. This is due to the expansion of side information including the increment in the length of the set of trimmed q's and expansion of the table. Hence, all available data embedding venues are utilized to accommodate the side information. This poor performance at $L>3$ is generally observed for all test signals considered. Thus, based on the results, we conclude that the optimum length for L is 3 bits where the maximum C is achieved for all types of signal. However, we emphasize that this conclusion is limited by applying GRC's.

Table 1 presents the effective embedding capacity (C) obtained by applying urDEED to various encrypted signals. Here, the results are collected by using $L=3$. Generally, in the case of image, C is in the range of $[334709, 401431]$ bits. Such variation in C is due to the differences in statistical features resulting from the application of various encryption algorithms to the image. Table 1 also shows the effective embedding rate in terms of *bit per bit* (bpb), which is defined as the ratio of C to the original size of the encrypted signal. It is noticed that the effective embedding rate is between 0.167 and 0.169 bpb. In the case of audio, C is in the range of $[472252, 518858]$ bits or in other words, between 0.167 and 0.184 bpb. The results are consistently close to that of applying urDEED to image.

Similar trend is also observed in the case of text, in which two files of different sizes are considered. The consideration of different sizes is to verify the effect of the original size on the performance of urDEED. It is noticed that when the smaller size is considered, C is in the range of $[17630, 19706]$ bits. For the larger size, C is in the range of $[400309, 401370]$ bits. Hence, C is proportional to the size of the input signal because C depends on λ_L, which depends on the total number of bits N as shown in Eq. (1). Nevertheless, the effective embedding rate is approximately the same in both cases. In the case of the smaller file-size document, the effective embedding rate ranges from 0.169 to 0.189 bpb, while this range is between 0.167 and 0.168 bpb in the case of the larger file-size document. This suggests that urDEED offers a consistent effective embedding rate regardless of the type of media and the encryption algorithm in use. These results also suggest that urDEED is able to embed external information into an encrypted signal without causing file-size increment nor data loss.

The second column of Table 2 compares the average effective embedding rate between Zhang's method [9] and the proposed urDEED for two images (i.e., Lenna and Man). Here, the average effective embedding rate of [9] is 0.006 bpb, which is based on the best reported results in [9] at the expense of sacrificing the reversibility property (i.e., perfect reconstruction is not possible at this embedding rate). However, urDEED achieves, on an average,

0.173 bpb, which outperforms the embedding capacity of Zhang's [9] by 28.8% and urDEED is completely reversible.

5.4. File-size preserving

File-size preserving is a crucial property in data embedding because in the scenario where the storage medium is limited in capacity (e.g., 4.7 GB for a single-layer DVD, a USB flash memory) or costly (e.g., networking), an expansion in file-size caused by data embedding may require additional storage medium or higher network traffic. Here, it is verified that the input signal (ciphertext) and the processed signal (chiphertext with extra payload embedded) are of the exact same file-size. This is an expected outcome because urDEED trims the quotient part of each GRC and embeds them along with the payload as detailed in subsection 3.3. Although Zhang's method [9,11] and its improvement [10] also preserve the file-size to that of the input encrypted image, it is clear that the proposed urDEED offers more features. These features include higher effective embedding capacity, applicability across different media, applicability across different encryption algorithms, and complete reversible functionality. Table 2 summarizes the functional comparison between Zhang's method and the proposed urDEED.

6. Conclusions

A new framework for embedding data in encrypted signal was proposed. This framework includes the mapping of input signal to imaginary codewords, followed by entropy coding and modifying them for locating venue to realize data embedding. urDEED is an application of such framework using Golomb–Rice codeword, and it achieved the following properties: (1) universally applicable to any encrypted signal and operational solely in the encrypted domain; (2) completely reversible, and; (3) file-size preserving while hosting external information in the encrypted signal. Results showed that an effective embedding rate of ~0.169 bpb is achieved regardless of the input media type and the encryption scheme in use.

As future work, we would like to increase the embedding capacity of urDEED by applying it to design hybrid data embedding method. Also, we want to apply the proposed framework by using other entropy coding methods such as Huffman and Arithmetic coding to realize data embedding in encrypted domain.

Acknowledgements

The authors would like to thank the anonymous reviewers for their constructive and invaluable comments in improving the quality of this article.

References

[1] B.Y. Lei, I.Y. Soon, Z. Li, Blind and robust audio watermarking scheme based on SVD-DCT, Signal Processing 91 (2011) 1973–1984.
[2] A. Cheddad, J. Condell, K. Curran, P.M. Kevitt, Digital image steganography: survey and analysis of current methods, Signal Processing 90 (2010) 727–752.
[3] D. Xu, R. Wang, J. Wang, A novel watermarking scheme for H.264/AVC video authentication, Signal Processing: Image Communication 26 (2011) 267–279.
[4] F. Lusson, K. Bailey, M. Leeney, K. Curran, A novel approach to digital watermarking, exploiting colour spaces, Signal Processing 93 (2013) 1268–1294.
[5] K. Wong, X. Qi, K. Tanaka, A DCT-based Mod4 steganographic method, Signal Processing 87 (2007) 1251–1263.
[6] B. Zhao, W. Kou, H. Li, L. Dang, J. Zhang, Effective watermarking scheme in the encrypted domain for buyer-seller watermarking protocol, Information Sciences 180 (2010) 4672–4684.
[7] S. Lian, Z. Liu, Z. Ren, H. Wang, Commutative encryption and watermarking in video compression, IEEE Transactions on Circuits and Systems for Video Technology 17 (2007) 774–778.
[8] M. Cancellaro, F. Battisti, M. Carli, G. Boato, F.D. Natale, A. Neri, A commutative digital image watermarking and encryption method in the tree structured haar transform domain, Signal Processing: Image Communication 26 (2011) 1–12.
[9] X. Zhang, Separable reversible data hiding in encrypted image, IEEE Transactions on Information Forensics and Security 7 (2012) 826–832.
[10] W. Hong, T.-S. Chen, H.-Y. Wu, An improved reversible data hiding in encrypted images using side match, IEEE Signal Processing Letters 19 (2012) 199–202.
[11] X. Zhang, Reversible data hiding in encrypted image, IEEE Signal Processing Letters 18 (2011) 255–258.
[12] C.H. Kim, Improved differential fault analysis on AES key schedule, IEEE Transactions on Information Forensics and Security 7 (2012) 41–50.
[13] R.C. Merkle, M.E. Hellman, On the security of multiple encryption, Communications of the ACM 24 (1981) 465–467.
[14] A. Moussa, Data encryption performance based on blowfish, in: 47th International Symposium ELMAR, 2005, pp. 131–134.
[15] S.V. Vaseghi, Advanced Digital Signal Processing and Noise Reduction, John Wiley & Sons, 2006.
[16] M. Rabbani, P.W. Jones, Digital image compression techniques, in: Society of Photo-Optical Instrumentation Engineers (SPIE), 1st edition, Bellingham, WA, USA, 1991.
[17] M. Weinberger, G. Seroussi, G. Sapiro, The LOCO-I lossless image compression algorithm: principles and standardization into JPEG-LS, IEEE Transactions on Image Processing 9 (2000) 1309–1324.
[18] G. Gogniat, T. Wolf, W. Burleson, J.-P. Diguet, L. Bossuet, R. Vaslin, Reconfigurable hardware for high-security/high-performance embedded systems: the safes perspective, IEEE Transactions on Very Large Scale Integration (VLSI) Systems 16 (2008) 144–155.
[19] M. McLoone, J. McCanny, High-performance FPGA implementation of DES using a novel method for implementing the key schedule, IEE Proceedings – Circuits, Devices and Systems 150 (2003) 373–378.
[20] B. Najafi, B. Sadeghian, M. Saheb Zamani, A. Valizadeh, High speed implementation of serpent algorithm, in: The 16th International Conference on Microelectronics, 2004, pp. 718–721.
[21] S.-L. Su, L.-C. Wuu, J.-W. Jhang, A new 256-bits block cipher twofish256, in: International Conference on Computer Engineering Systems, 2007, pp. 166–171.
[22] H.-M. Sun, M.-E. Wu, W.-C. Ting, M. Hinek, Dual RSA and its security analysis, IEEE Transactions on Information Theory 53 (2007) 2922–2933.
[23] C. Adams, H. Heys, S. Tavares, M. Wiener, An analysis of the CAST-256 cipher, in: IEEE Canadian Conference on Electrical and Computer Engineering, vol. 1, 1999, pp. 361–366.
[24] B.V. Rompay, L.R. Knudsen, V. Rijmen, Differential cryptanalysis of the ICE encryption algorithm, in: Proceedings of the 5th International Workshop on Fast Software Encryption, FSE '98, 1998, pp. 270–283.
[25] E.M. Mohamed, S. El-Etriby, H.S. Abdul-kader, Randomness testing of modern encryption techniques in cloud environment, in: 8th International Conference on Informatics and Systems (INFOS), 2012, pp. CC-1 –CC-6.
[26] F. Wen, L. Yin, Optimal tweakable blockcipher based on dual MISTY-type structure, in: Second International Conference on Future Networks, 2010, pp. 39–41.
[27] D. Moon, K. Hwang, W. Lee, S. Lee, J. Lim, Impossible differential cryptanalysis of reduced round XTEA and TEA, in: Revised Papers from the 9th International Workshop on Fast Software Encryption, FSE '02, 2002, pp. 49–60.