# A Generic Blockchain-based Steganography Framework with High Capacity via Reversible GAN

Zhuo Chen†, Liehuang Zhu†, Peng Jiang†, Jialing He‡, Zijian Zhang†
† School of Cyberspace Science and Technology, Beijing Institute of Technology, China
‡ College of Computer Science, Chongqing University, China
Email: {chen.zhuo, liehuangz}@bit.edu.cn, pennyjiang0301@gmail.com, hejialing@cqu.edu.cn, zhangzijian@bit.edu.cn

IEEE INFOCOM 2024 - IEEE Conference on Computer Communications | 979-8-3503-8350-8/24/$31.00 ©2024 IEEE | DOI: 10.1109/INFOCOM52122.2024.10621377

***Abstract*—Blockchain-based steganography enables data hiding via encoding the covert data into a specific blockchain transaction field. However, previous works focus on the specific field-embedding methods while lacking a consideration on required field-generation embedding. In this paper, we propose GBSF, a generic framework for blockchain-based steganography. The sender generates the required fields, where the additional covert data is embedded to enhance the channel capacity. Based on GBSF, we design R-GAN that utilizes the generative adversarial network (GAN) with a reversible generator to generate the required fields and encode additional covert data into the input noise of the reversible generator. We then explore the performance flaw of R-GAN and introduce CCR-GAN as an improvement. CCR-GAN employs a counter-intuitive data preprocessing mechanism to reduce decoding errors in covert data. It incurs gradient explosion for model convergence and we design a custom activation function. We conduct experiments using the transaction amount of the Bitcoin mainnet as the required field. The results demonstrate that R-GAN and CCR-GAN allow to embed 11-bit (embedding rate of 17.2%) and 24-bit (embedding rate of 37.5%) covert data within a transaction amount, and enhance the channel capacity of state-of-the-art works by 4.30% to 91.67% and 9.38% to 200.00%, respectively.**

***Index Terms*—blockchain, steganography, covert channel, capacity enhancement, GAN**

## I. Introduction

Steganography enables both senders and receivers to transmit data secretly over a public network channel [1]–[3]. It is widely used in digital watermarking [4], censorship-resistant systems [5] and digital forensics [6]. Due to concealing the communication behavior between the sender and the receiver, steganography ensures a secure transmission of confidential military and commercial information [7]. In the blockchain-based steganography [8], [9], the sender and the receiver establish a covert channel through the blockchain network [10], [11]. The sender hides the covert data into a specific transaction field and broadcasts the covert transaction to the blockchain. The receiver identifies the covert transaction from the blockchain and decodes it to access the covert data.

Previous achievements primarily focus on encoding the covert data into specific transaction fields (i.e., embedding fields), while rarely considering the proper generation of required fields that used to complete a transaction [12], [13]. Wang et al. [14] demonstrated that the improper generation of required fields can easily expose covert transactions, and proposed a required-field-generation method by applying generative adversarial networks (GAN) [15]. This approach achieves required fields that are indistinguishable from normal transaction fields and limited capacity. It lacks a consideration of embedding the covert data into the indistinguishable required fields' generation process, which mainly has the following challenges.

- *Less redundancy.* Blockchain fields contain less redundant information compared to audio and image data [16], [17], making it more challenging to conceal data.
- *No semantics.* Unlike text, a blockchain field is simply a number without semantic information, rendering mainstream text steganography methods [18]–[21], which rely on semantic and state transfer probabilities, unsuitable for blockchain field steganography.
- *Difficult to encode.* Existing studies often employ deep generative models to generate indistinguishable required fields [14], while these models are often uninterpretable, making it difficult to encode data into the generated fields.

In this paper, we propose GBSF, a generic blockchain-based steganography framework that boosts the capacity of blockchain-based covert channels. In GBSF, the sender employs GAN to generate required fields and encodes the covert data as input to GAN's generator. This input consists of random noise, which is analogous to the random string nature of covert data (often ciphertext). However, it presents a new challenge for the receiver, as deep learning models are typically irreversible, making it difficult to restore covert data. To address this challenge, we introduce the concept of reversible GAN whose generator is reversible. Based on the concept, we instantiate the R-GAN scheme. Our key insight is to model the generator of R-GAN as an invertible multivariate function by carefully configuring the network structure. This is accomplished by constructing the generator using linear neural networks (e.g., fully connected layers and convolutional neural networks) and reversible activation functions (e.g., Sigmoid and LeakyReLU). The receiver reverses the generator to retrieve the input noise and the covert data.

To enhance performance, we introduce CCR-GAN which improves R-GAN with a counter-intuitive data preprocessing method and a custom activation function. R-GAN is limited

This work is supported by the National Defense Basic Scientific Research program of China under grant number JCKY2020602B008.
Corresponding authors: Peng Jiang and Zijian Zhang.

to recovering the covert data due to the generator's tendency to produce fractional numbers, whereas the transaction field requires integer values. Consequently, it incurs inaccuracy and computational errors. We propose a counter-intuitive preprocessing method (CIDP) to mitigate the limitation. However, CIDP incurs gradient explosion for model convergence. We design a custom activation function called ClipSigmoid as the output of the model to eliminate excessive gradient.

Finally, we launch an implementation of both R-GAN and CCR-GAN on the Bitcoin mainnet from capacity and concealment, using the transaction amount as a required field. The experimental results reveal that R-GAN allows to embed 11-bit covert data in each transaction amount (embedding rate of 17.2%), with a capacity increase of state-of-the-art works by 4.30% to 91.67%. With CCR-GAN, it becomes possible to embed 24-bit covert data in each transaction amount (embedding rate of 37.5%), increasing the capacity by 9.38%-200% compared with state-of-the-art works. The concealment of R-GAN and CCR-GAN are higher than baselines, with identification accuracies of approximately 0.67 and 0.73, respectively (the closer the accuracy is to 0.5, the higher the concealment). Both the capacity and concealment of the proposed schemes surpass those of the baselines that embed covert data into the transaction amount. Experimental results show that the proposed GBSF framework can enhance both the concealment and capacity of the blockchain-based steganography due to the use of the reversible GAN to embed additional covert data in the generation of required fields.

In summary, main contributions of this paper include:

- We propose a GBSF framework that enhances the capacity of blockchain-based covert channels. In GBSF, the sender leverages the reversible GAN to generate the required fields for creating transactions, while encoding covert data as the input to the generator.
- We instantiate two schemes, namely R-GAN and CCR-GAN. The main concept behind R-GAN is to model its generator as a reversible function that allows the receiver to decode covert data by reversing the generator's computation. However, R-GAN has a capacity limitation due to rounding errors. We further propose CCR-GAN to improve capacity. In comparison to R-GAN, CCR-GAN incorporates a counter-intuitive data preprocessing method named CIDP and a custom activation function called ClipSigmoid. These additions reduce rounding errors and facilitate model convergence. We also present formal justifications for CIDP and ClipSigmoid.
- We utilize the transaction amount as the required field for implementing GBSF on the Bitcoin mainnet. We conduct tests to measure the amount of covert data that R-GAN and CCR-GAN can embed in each transaction amount, as well as their capacity to enhance existing blockchain-based steganography schemes. Experimental results demonstrate that both schemes exhibit a high level of concealment and capacity enhancement capabilities.

## II. GBSF Framework

We introduce a generic framework for enhancing the capacity of blockchain-based steganography, as illustrated in Fig. 1. The framework consists of an original covert channel and an expanding covert channel.

The blue box illustrates the original covert channel, consisting of a sender, a receiver, and a blockchain. The sender encodes the covert data into a transaction field (i.e., the embedding field) such as the address and the signature. Afterwards, the sender generates the remaining transaction fields required for transaction creation (i.e., the required field), either randomly or using deep generative models. These required fields are utilized to create the covert transaction, which is then broadcasted to the blockchain network. The receiver retrieves the covert transaction from the blockchain and decodes the covert data based on the embedding field.

The expanding covert channel (depicted in the yellow box in Fig. 1) enhances the channel capacity, which is achieved by embedding additional covert data in required fields rather than generating required fields randomly or using deep generative models. The key concept behind the expanding covert channel is the utilization of a specialized GAN called reversible GAN. Similar to a typical GAN, the reversible GAN consists of a generator and a discriminator. The difference is that its generator is reversible, allowing the function expression of the generator to be reversed. This property enables the sender to encode additional covert data using the generator in the forward direction, while the receiver can utilize the generator in the reverse direction to decode the additional covert data. With the reversible GAN, the sender can encode additional covert data into transaction fields that are indistinguishable from normal transaction fields. These fields carrying the additional covert data are known as expansion fields. By incorporating the expansion fields alongside the original embedding fields, the sender constructs the complete covert transactions, effectively increasing the capacity of the blockchain-based covert channel. The receiver retrieves the expansion field from the covert transaction and inputs it into the reversible GAN. The reversible GAN then performs the reverse process of encoding and outputs the additional covert data. In this way, we establish an expanding covert channel to transmit the additional covert data and boost the channel capacity.

## III. Expanding Covert Channel with R-GAN

### A. Overview

As depicted in Fig. 2, the expanding covert channel consists of three main steps. Firstly, the sender and the receiver train a model to generate expansion fields. Secondly, the sender encodes covert data into the generated expansion fields. Finally, the receiver decodes the expansion fields to recover the covert data. Note that this paper focuses on details of the encoding and decoding principle and does not consider the model synchronization between the sender and the receiver. The sender and the receiver can simply obtain an identical model by sharing the rules for training the model. For example,

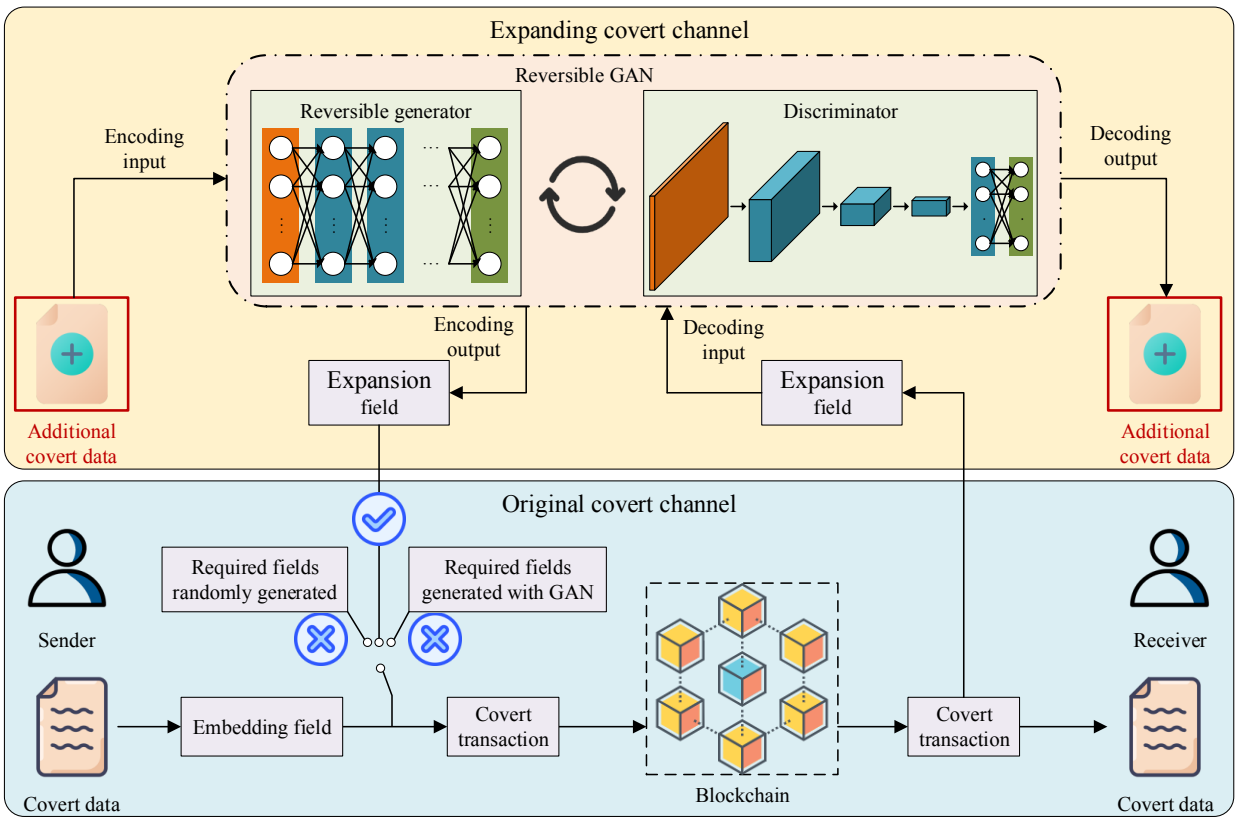



Fig. 1: Overview of GBSF framework.

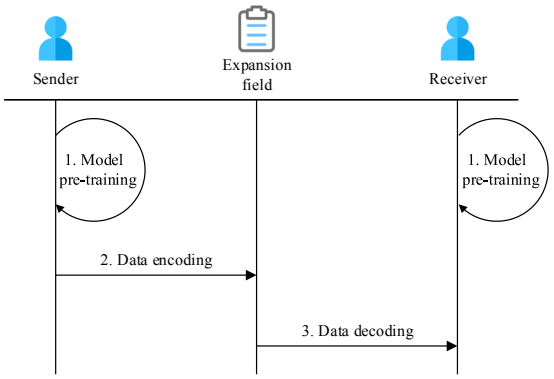
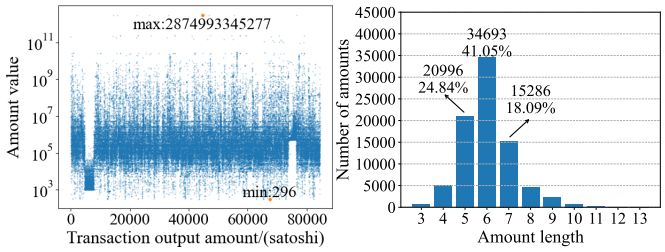



Fig. 2: General workflow of the expanding covert channel.




(a) Transaction amount. (b) Length of transaction amount.

Fig. 3: Example of Bitcoin transaction amount.

every 3 days, they select the expansion field from the last 100 blocks as a dataset to train the model using a series of identical seeds. The training process does not stop until the model loss falls below a certain threshold.

### B. Model Pre-training

We utilize the transaction output amount of Bitcoin as the expansion field to illustrate the pre-training process. We specifically choose this amount field since each transaction must include a specified output amount and the transaction creator has full control over the output amount.

*1) Data preprocessing:* The model takes the transaction output amount as the input, which is represented as an integer like "18105990". Fig. 3(a) illustrates the distribution of Bitcoin transaction output amounts for 25 blocks, from block 727215 to block 727239. These amounts range from 296 to 2,874,993,345,277, with a dense distribution between $10^5$ and $10^7$. Fig. 3(b) displays the number and proportion of amounts grouped by length. Amounts with lengths ranging from 5 to 7 account for a significant portion, totaling 83.98%. We select the most common data, specifically data with lengths ranging from 5 to 7, as the training dataset $X$, and apply min-max normalization to normalize the training dataset to the range from 0 to 1. Assume $X = [x_1, x_2, \cdots, x_n]$, then for $i \in [1, 2, \cdots, n]$, we normalize

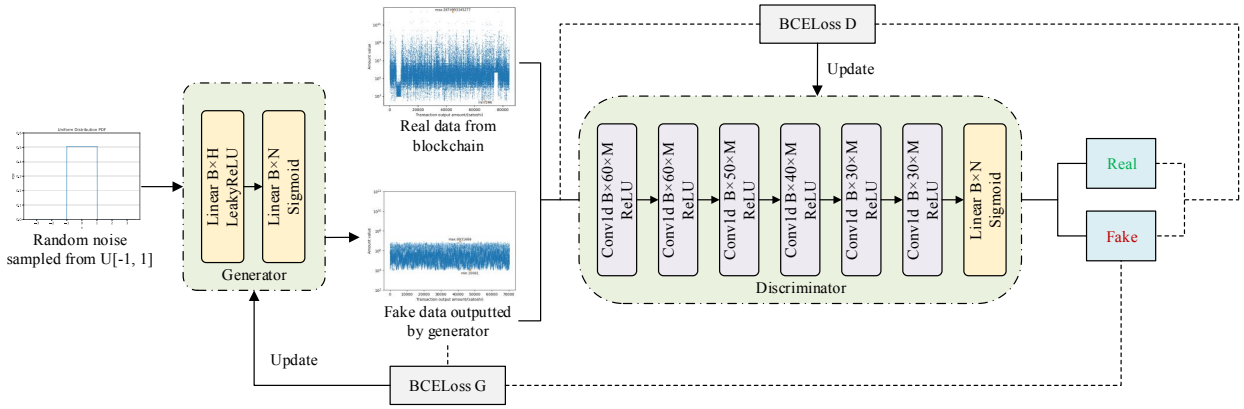



Fig. 4: Overview of R-GAN. $U$ refers to a uniform distribution; $B$ is the batchsize; $H$ represents the hidden dimension; $M$ denotes the input dimension; $N$ is the output dimension. BCELoss refers to the binary cross entropy loss.

$x'_i = \frac{x_i - min(X)}{max(X) - min(X)}$, where $min(X)$ and $max(X)$ represent the minimum and maximum elements of $X$. We also reshape the dataset and group adjacent 64 data points to form a new training dataset. The formation process is represented as $new\_x_i = [x'_{(i-1)*64+1}, x'_{(i-1)*64+2}, \cdots, x'_{i*64}]$, where $i \in [1, 2, \cdots, \lceil n/64 \rceil]$.

*2) Model structure:* Fig. 4 presents the overview of R-GAN, comprising a generator and a discriminator. The generator takes a random noise sampled from $U[-1, 1]$ (uniform distribution) as the input and fake amounts as the output. We use a uniform distribution to sample the input noise to accommodate covert data within the noise. The covert data (typically encrypted) encoded into the generator's input can be seen as a random string, which exhibits randomness as well as uniform distribution. The generator consists of two fully connected layers to ensure reversibility. While the fully connected layer is a simple neural network, it is sufficient to handle one-dimensional numerical data like the transaction amount. The first layer utilizes LeakyReLU as the activation function due to its piecewise linearity property, which can reduces decoding errors during covert data recovery. The last layer employs Sigmoid as the activation function, as it is a monotonic function that outputs values between 0 and 1. The monotonicity of Sigmoid enables reversibility, and its output range aligns with normalized training data. The discriminator focuses on evaluating the concealment of generated fake data and is unrelated to the recovery of the covert data. Complex irreversible networks can be employed to construct the discriminator. Due to the excellent feature capture capabilities of convolutional neural networks (CNNs), we use 6 CNNs with ReLU to form the discriminator. The last layer of the discriminator is a fully connected layer with Sigmoid, commonly utilized for binary classification tasks.

*3) Loss function:* We use the binary cross-entropy loss (BCELoss) to calculate the loss function since BCELoss is more suitable for handling values between 0 and 1 and processing binary classification tasks. The loss function of the model comprises two parts, including the discriminator's loss and the generator's loss. The discriminator's loss is defined as the prediction error of the discriminator on real and fake samples. Assume that the label for real samples is 1 and the label for fake samples is 0, then the loss function of the discriminator can be expressed as $J_D = -\frac{1}{2}\sum(\log(1 - D(G(noise))) + \log D(x'))$. Where $D$ represents the discriminator, $G$ is the generator, $noise$ denotes the input noise of $G$, and $x'$ represents the normalized element of the training dataset $X$. The second part penalizes the generator for generating fake samples that are recognized by the discriminator, which we measure by $J_G = -\sum \log D(G(noise))$.

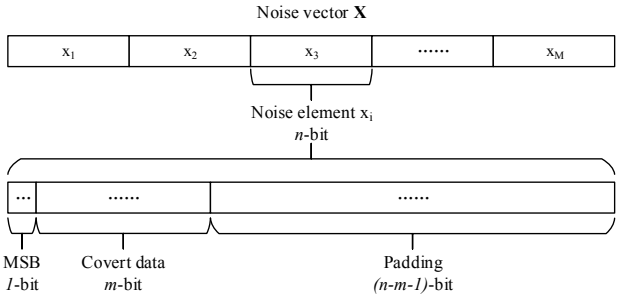



Fig. 5: Noise structure. MSB is the most significant bit.

## C. Data Encoding

The data encoding process consists of two phases: embedding and verification. In the embedding phase, covert data is embedded into the noise, while the verification phase ensures that the receiver can correctly recover the covert data in the presence of the rounding and computational errors.

**Embedding phase.** The sender incorporates the covert data into the input of the generator, which is a noise vector. The sender replaces certain bits of each noise element with covert data. Fig. 5 illustrates the structure of the noise, which consists of 1-bit most significant bit (MSB), $m$-bit covert data, and $(n-m-1)$-bit Padding. Both MSB and Padding are uniformly sampled, ensuring the noise spans a wide range of values.

**Verification phase.** The verification phase ensures that the bits representing the covert data in the recovered noise match those

sent by the sender. This phase is necessary because the receiver may not obtain the exact same noise as the sender during the decoding process. The inconsistencies may arise due to computational errors and rounding errors. The computational errors refer to inaccuracies in decimal calculations on computers, while the rounding errors occur when the generator's decimal amounts are rounded to on-chain integer amounts.

For simplicity and better understanding, assume that the generator takes an $M$-dimensional noise vector $\mathbf{X}$ as input, where each element consists of $n$ bits. The covert data is represented by an $M$-dimensional vector $\mathbf{CD}$, with each element consisting of $m$ bits. We denote the generator as $G(\cdot)$ and its inverse as $G^{-1}(\cdot)$. The data encoding process is illustrated in Algorithm 1. The sender starts by sampling MSB and Padding for each covert data element $cd_i$, and concatenates them to form an $n$-bit noise element (lines 4-8). Using the noise vector composed of these noise elements, the sender obtains a decimal transaction amount vector $\mathbf{A}$ based on $G(\cdot)$ (lines 9-10). Since on-chain transaction amounts must be integers, decimal values are rounded to integer values $\hat{\mathbf{A}}$ (line 12). The sender then computes the recovered noise based on the rounded integer values (lines 13-14) and verifies whether the covert data bits in the recovered noise match those of the original noise (lines 15-26).The sender iteratively performs the

**Algorithm 1:** Data encoding.

**Input:** Covert data $\mathbf{CD} = (cd_1, cd_2, \cdots, cd_M) \in \{0,1\}^{m \times M}$.
**Output:** Noise $\mathbf{X} = (x_1, x_2, \cdots, x_M) \in \mathbb{R}^M$.
1 Initialize $\mathbf{X} = (x_1, x_2, \cdots, x_M)$;
2 **while** *True* **do**
3  // **Begin embedding phase**;
4  **for** $i$ *in* $range$(M) **do** Randomize $MSB$ and $Padding$
5  Uniformly sample an $MSB_i \in \{0,1\}$;
6  Uniformly sample a $Padding_i \in \{0,1\}^{n-m-1}$;
7  Set $x_i = MSB_i || cd_i || Padding_i$;
8  **end**
9  Compute $\mathbf{Y} = G(\mathbf{X})$;
10  Denormalize $\mathbf{A} = \mathbf{Y} \times (max(X) - min(X)) + min(X)$;
11  // **Begin verification phase**;
12  Round $\hat{\mathbf{A}} = [\mathbf{A}]$;
13  Normalize $\hat{\mathbf{Y}} = \frac{\hat{\mathbf{A}} - min(X)}{max(X) - min(X)}$;
14  Compute $\hat{\mathbf{X}} = G^{-1}(\hat{\mathbf{Y}})$;
15  Initialize $Result$ = [];
16  **for** $\hat{x}_i \in \hat{\mathbf{X}}$ **do**
17  **if** *bits* 2 *to* $(m+1)$ *of* $\hat{x}_i$ == *bits* 2 *to* $(m+1)$ *of* $x_i$ **then**
18  $Result.append(True)$;
19  **end**
20  **else**
21  $Result.append(False)$;
22  **end**
23  **end**
24  **if** *All elemets of* $Result$ *are True* **then**
25  Break;
26  **end**
27 **end**
28 **return** $\mathbf{X}$

embedding phase and the verification phase until a satisfying noise vector is obtained. Note that the infinite loop (line 2) can be prevented by randomizing MSB.

### D. Data Decoding

The main concept behind data decoding is to reverse the computation of the generator. Let the first layer has an $M$-dimensional input and an $H$-dimensional output, and the second layer has an $H$-dimensional input and an $N$-dimensional output. LeakyReLU has a slope of $\alpha$. Suppose the receiver obtains a transaction amount vector $\hat{\mathbf{A}} = (\hat{y}_1, \hat{y}_2, \cdots, \hat{y}_N) \in \mathbb{Z}_+^N$. The receiver can recover the covert data $\widehat{\mathbf{CD}} = (\widehat{cd}_1, \widehat{cd}_2, \cdots, \widehat{cd}_M) \in \{0,1\}^{m \times M}$ in the following steps outlined in Fig. 6.

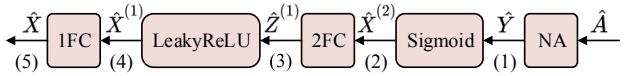



Fig. 6: Data decoding. 1FC and 2FC refers to the first fully connected layer and the second fully connected layer. NA denotes normalization.

(1) Normalize the transaction amount to get the output of Sigmoid. Given $\hat{\mathbf{A}}$, compute $\hat{\mathbf{Y}} = \frac{\hat{\mathbf{A}} - min(X)}{max(X) - min(X)}$, where $\hat{\mathbf{Y}} = (\hat{y}_1, \hat{y}_2, \cdots, \hat{y}_N) \in \mathbb{R}_+^N$.
(2) Calculate the output of the second layer. Given the output of the generator $\hat{\mathbf{Y}}$, compute $\hat{\mathbf{X}}^{(2)} = Logistic(\hat{\mathbf{Y}})$, where $Logistic(\cdot)$ is the logistic function and $\hat{\mathbf{X}}^{(2)} = (\hat{x}_1^{(2)}, \hat{x}_2^{(2)}, \cdots, \hat{x}_N^{(2)}) \in \mathbb{R}^N$.
(3) Calculate the output of LeakyReLU. Given the output of the second layer $\hat{\mathbf{X}}^{(2)}$, compute $\hat{\mathbf{Z}}^{(1)} = \mathbf{W}_2^{-1}\hat{\mathbf{X}}^{(2)}$, where $\mathbf{W}_2^{-1}$ is the inverse of the second layer's weight matrix and $\hat{\mathbf{Z}}^{(1)} = (\hat{z}_1^{(1)}, \hat{z}_2^{(1)}, \cdots, \hat{z}_H^{(1)}) \in \mathbb{R}^H$. Note that $\mathbf{W}_2$ is a matrix with $N$-row and $H$-column , and $\mathbf{W}_2^{-1}$ exists only when $N = H$ and $\mathbf{W}_2$ is a full-rank matrix. We thus set $N = H$ in the model training process.
(4) Calculate the output of the first layer. Given the output of LeakyReLU $\hat{\mathbf{Z}}^{(1)} = (\hat{z}_1^{(1)}, \hat{z}_2^{(1)}, \cdots, \hat{z}_H^{(1)}) \in \mathbb{R}^H$, for each $\hat{z}_i^{(1)}$ where $i \in [1, 2, \cdots, H]$, compute $\hat{x}_i^{(1)} = \hat{z}_i^{(1)}$ for which $\hat{z}_i^{(1)} \geq 0$ and $\hat{x}_i^{(1)} = \frac{1}{\alpha}\hat{z}_i^{(1)}$ for which $\hat{z}_i^{(1)} < 0$, and set $\hat{\mathbf{X}}^{(1)} = (\hat{x}_1^{(1)}, \hat{x}_2^{(1)}, \cdots, \hat{x}_H^{(1)}) \in \mathbb{R}^H$.
(5) Calculate the recovered noise $\hat{\mathbf{X}}$. Given $\hat{\mathbf{X}}^{(1)}$, compute $\hat{\mathbf{X}} = \mathbf{W}_1^{-1}\hat{\mathbf{X}}^{(1)}$, where $\mathbf{W}_1^{-1}$ is the inverse of the first layer's weight matrix and $\hat{\mathbf{X}} = (\hat{x}_1, \hat{x}_2, \cdots, \hat{x}_M) \in \mathbb{R}^M$. $\mathbf{W}_1$ is a matrix with $H$-row and $M$-column. We also set $H = M$ such that $\mathbf{W}_1^{-1}$ exists.
(6) Intercept and concatenate the recovered noise's covert data bits. Given $\hat{\mathbf{X}} = (\hat{x}_1, \hat{x}_2, \cdots, \hat{x}_M) \in \mathbb{R}^M$, compute $\widehat{\mathbf{CD}} = (\hat{x}_1[2:m+1], \cdots, \hat{x}_M[2:m+1]) \in \{0,1\}^{m \times M}$, where $\hat{x}_i[2:m+1]$ refers to the bit string formed by concatenating the 2nd bit to the $(m+1)$th bit of $\hat{x}_i$.

Experiments in Section V-B demonstrate that R-GAN permits embedding 11-bit covert data per transaction amount. To enhance performance, we propose a counter-intuitive data preprocessing module and a custom activation function to increase the amount of covert data that can be embedded in each transaction amount.

## IV. CCR-GAN

In this section, we propose CCR-GAN to improve R-GAN. The limited performance of R-GAN stems from the

discrepancy between the noise recovered by the receiver and that sent by the sender, which arises from two errors.

- The first error is the precision error inherent to computers when performing decimal calculations, as there exists a built-in precision limit.
- The second error occurs when rounding transaction amounts. The generator of R-GAN generates a decimal value, while on-chain transaction amounts must be integers. The sender rounds the decimal number to an integer, which introduces errors.

The first error can be reduced by utilizing data types with higher precision. To reduce the second type of error, we initially introduce a counter-intuitive data preprocessing mechanism referred to as CIDP.

**CIDP.** Recall that the sender selects data with a higher occurrence probability as the training dataset. This choice allows the model to disregard extreme data and their influence on feature extraction, promoting generating data closely resembling normal data. However, we observe that the selection also leads to an increase in rounding errors. This is because the smaller the multiplier (i.e., $max(X) - min(X)$) in denormalization, the relatively larger the value being rounded, and the larger the rounding error. We propose CIDP, in which the sender includes all data as part of the training dataset, to decrease the error. By doing so, extremely large and small data points are not excluded, resulting in an increased $max(X) - min(X)$ and, consequently, an decreased rounding error. However, not selecting data incurs an extremely uneven distribution of normalized training data. About $83.98\%$ of the data falls between $10^{-8}$ and $10^{-5}$, while the entire range is from $0$ to $1$. This unevenness makes the model difficult to converge.

**ClipSigmoid.** The convergence of R-GAN with CIDP is extremely difficult since the weight update gradient during backpropagation is too large relative to the input data. It leads to difficulties in achieving fine-grained variations in the model's output. Each step taken by the model in the search for an optimal solution is too large to reach a better solution. An intuitive solution is to modify the learning rate to mitigate the influence of weight updates on the model's outcomes. However, it is challenging in practice because determining when to decrease the learning rate and by how much is not straightforward. Furthermore, a very low learning rate can significantly prolong the training process.

To this end, we introduce a custom activation function called ClipSigmoid. Its main concept is to suppress the gradient during backpropagation when the model generates small data, which is achieved by setting the gradient of a specific segment of Sigmoid to zero. We define ClipSigmoid as follows:

$$ClipSigmoid(x) = \begin{cases} 1e^{-20}, Sigmoid(x) \leq 1e-20 \\ Sigmoid(x), Sigmoid(x) > 1e-20. \end{cases}$$

Fig. 7 illustrates the distinction between ClipSigmoid and Sigmoid. ClipSigmoid sets the lowest threshold for Sigmoid. When the value of Sigmoid falls below this threshold, ClipSigmoid enforces the value to this threshold. In our case, we set this threshold to $1e-20$, which is a hyperparameter established during the model training process. Whenever the model generates a number below this threshold, the gradient of the weight update in the backpropagation step becomes zero. This effectively prevents the model from pursuing an incorrect solution by eliminating further exploration in that direction.

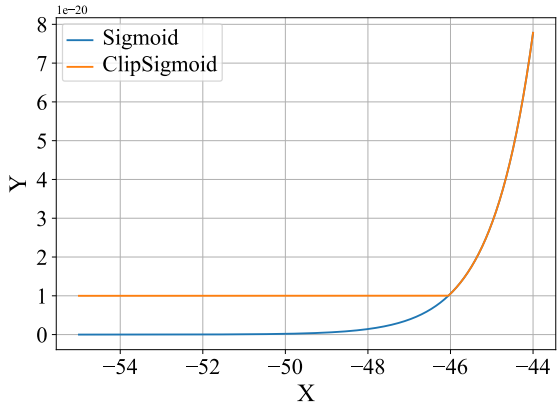


Fig. 7: Different part of ClipSigmoid and Sigmoid.

Note that the activation function is an integral part of the generator. It is thus necessary for ClipSigmoid to be reversible, allowing the receiver to compute the corresponding input $x$ from the model's output $y$. Although the zero-segment of ClipSigmoid is an irreversible straight line parallel to the $x$-axis, it does not impact the reversibility of R-GAN. The sender can intentionally train the generator to yield data outside the zero-segment to avoid the irreversible segment. ClipSigmoid's effectiveness is presented in Section V-B3.

## V. Experiments

We evaluate R-GAN and CCR-GAN by using the output amount of Bitcoin as the expansion field since most schemes do not utilize the transacton amount as the embedding field. This implies that amount-based (CC)R-GAN can significantly enhance the capacity of most existing solutions. For a few schemes that already consider the output amount as the embedded field, (CC)R-GAN can still increase their capacity by utilizing other numeric fields such as fees and gas as expansion fields. Note that we do not conduct additional experiments on these numeric fields because experiments on the transaction amount already demonstrate the feasibility of our schemes. Necessarily, we compare (CC)R-GAN with the existing scheme that uses the Bitcoin amount as the embedding field to show superiority.

### A. Setup

Both R-GAN and CCR-GAN are implemented using Python and PyTorch 1.13.1+cpu, and trained on machines with an Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz 1.80 GHz and 8.00 GB RAM. The training process is very fast and can be completed in a few minutes on the CPU.

*1) Dataset:* We collect a dataset including 84,515 output amounts from Bitcoin transactions downloaded from block 727215 to block 727239 in the Bitcoin mainnet. These amounts are measured in satoshi and are represented as integers ranging from 296 to $2,874,993,345,277$. In CCR-GAN, we discard amounts that cannot form a complete batch of input data based on the batch size and the input dimension. In the case of R-GAN, we include amounts between $10^5$ and $10^8-1$ as the training set, and also discard redundant amounts.

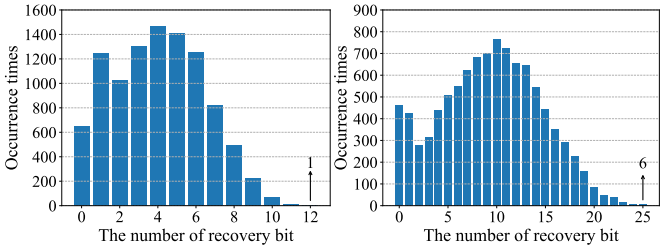
*2) Baselines:* We make comparisions between our schemes and four baselines that use fields other than the output amount as the embedding field to evaluate the channel expansion capacity: **(1) HC-CDE** [22] encodes covert data using the computational relationship between transaction addresses. **(2) DSA** [23] includes schemes that replace random factors in the signing process with covert data. **(3) Un-UTXO** [24] encodes covert data as an transaction output address. **(4) DLchain** [25] utilizes the private key as the carrier of covert data. These baselines are widely recognized for high concealment, which makes them suitable for comparison. In addition, we compare the capacity and concealment against **CCMBBT** [26], which uses the Bitcoin amount as the embedding field.

*3) Performance metrics:* We consider three performance metrics: **(1) Absolute capacity (AC)** indicates the amount of data that can be embedded in each expansion field. It is defined as $\mathsf{AC} = \frac{n}{N_f}\ \ bit$, where $N_f$ is the number of expansion fields, and $n$ is the number of bits of covert data carried by these expansion fields. **(2) Capacity expansion rate (CER)** refers to the capacity by which the expansion covert channel enhances the original covert channel. It is computed by $\mathsf{CEC} = \frac{\mathsf{AC}_{ecc}}{\mathsf{AC}_{occ}} \times 100\%$, where $\mathsf{AC}_{ecc}$ represents AC of the expansion covert channel, and $\mathsf{AC}_{occ}$ represents AC of the original covert channel. **(3) Concealment.** Concealment refers to the ability of embedding/expansion fields to remain undetected. We evaluate the concealment using the accuracy, precision, recall, and F1_score of CTR model [14], which is a steganalysis model used to detect the presence of covert data within a transaction field. A concealment score closer to 0.5 indicates a higher level of concealment.

*4) Parameter setting:* The batch size is set to 10, and the input/hidden/output dimensions are set to 64. LeakyReLU are with $\alpha = 0.3$. The learning rate of both R-GAN and CCR-GAN is set to 0.001, while the learning rate of CCR-GAN decreases to 0.9 times the original value after every two epochs. CCR-GAN stops training when the loss does not decrease in 10 consecutive epochs, while R-GAN stops training when that does not decrease in 5 consecutive epochs. The amount values are normalized using the maximum-minimum normalization method. All parameters, including weights, biases, and input/output data, are of type float64 to minimize the computational precision error.

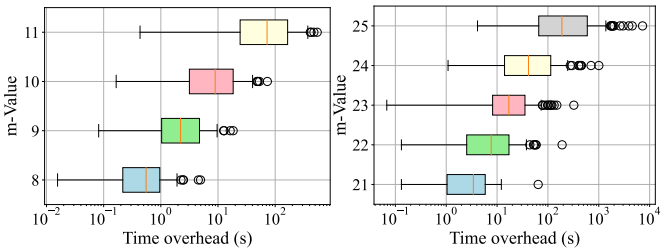
### *B. Self-comparison Experiments*

We compare AC and concealment of R-GAN and CCR-GAN, respectively. Recall that only $m$ bits in each noise element are used to store covert data. We begin by conducting experiments to determine the appropriate value of $m$ and then evaluate the performance.




(a) Recovery bits for R-GAN. (b) Recovery bits for CCR-GAN.

Fig. 8: Experiments on recovery bits (including 1-bit MSB).




(a) Test $m$ for R-GAN. (b) Test $m$ for CCR-GAN.

Fig. 9: Time overhead to find a satisfying noise vector.

*1) Experiments on* AC*:* **We first determine the approximate value of $m$.** We calculate $m = \min(NPID_2(\hat{x}_i, x_i))-1$, where $i \in [1, 2, \cdots, M]$ where $NPID_2$ denotes binary calculation. The first $m+1$ bits of each element at the corresponding position in $\hat{\mathbf{X}}$ and $\mathbf{X}$ should be identical to ensure accurate recovery of the $m$-bit covert data by the receiver. When obtaining a trained model, we need to measure the approximate $m$ in the following manner.

Given a trained model, we input a completely random noise $\mathbf{X}$, calculate the reduced noise $\hat{\mathbf{X}}$ obtained by the receiver, and compute the minimum recovery bit as $min(NPID_2(\hat{x}_i, x_i))$. To determine the appropriate range of $m$, we repeat the above steps 10,000 times. Experimental results are presented in Fig. 8. We focus on the highest number of recovered bits. For R-GAN, 1 out of 10,000 experiments is able to recover 12 bits of noise. For CCR-GAN, 6 out of 10,000 experiments can recover 25 bits of noise. The results indicate that R-GAN and CCR-GAN have the potential to allow the receiver to recover approximately 11 and 25 bits of covert data, respectively. Meanwhile, the sender must consider the computational resources and time required during the data encoding's verification phase. We determine the accurate value of $m$ in subsequent experiments based on the time consumption.

**We then determine accurate values of $m$ for R-GAN and CCR-GAN when embedding certain covert data, respectively.** When embedding certain covert data, the variability of the noise is reduced compared to the previous experiments. The covert data is embedded in bits 2 to $m+1$ of the noise, meaning these $m$ bits are fixed and cannot be randomized.

TABLE I: Time overhead required to find a satisfying noise and noise vector for R-GAN and CCR-GAN at different $m$.

| Scheme | $m$ | Time overhead for a vector (s) | | | | Time overhead for an amount (s) | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | 1st Quartile | 2nd Quartile | 3rd Quartile | Average | 1st Quartile | 2nd Quartile | 3rd Quartile | Average |
| R-GAN | 8 | 0.216 | 0.552 | 0.959 | 0.736 | 0.003 | 0.003 | 0.015 | 0.011 |
| | 9 | 1.033 | 2.215 | 4.748 | 3.405 | 0.016 | 0.035 | 0.074 | 0.053 |
| | 10 | 3.160 | 8.934 | 18.430 | 13.215 | 0.049 | 0.140 | 0.288 | 0.206 |
| | 11 | 24.365 | 72.127 | **165.125** | 119.866 | 0.381 | 1.127 | **2.580** | 1.873 |
| CCR-GAN | 21 | 1.038 | 3.419 | 5.852 | 4.385 | 0.016 | 0.053 | 0.091 | 0.068 |
| | 22 | 2.527 | 7.677 | 17.155 | 13.766 | 0.039 | 0.120 | 0.268 | 0.215 |
| | 23 | 8.170 | 17.055 | 35.379 | 31.931 | 0.128 | 0.266 | 0.553 | 0.499 |
| | 24 | 14.022 | 41.502 | **113.695** | 100.168 | 0.219 | 0.648 | **1.776** | 1.565 |
| | 25 | 65.929 | 188.369 | *594.831* | 584.567 | 1.030 | 2.943 | *9.294* | 9.134 |

Furthermore, these $m$ bits occupy the more significant positions which have a greater influence on the overall noise value. When significant bits are fixed, it may take more time (or may not even be possible) to find a satisfactory noise vector. We select "The Little Prince" as transmitted data to align with CCMBBT, and the covert data is encrypted using AES-CBC.

In the case of R-GAN, we assess the time overhead of finding 100 noise vectors for different covert data where the values of $m$ ranging from 8 to 11. Fig. 9(a) illustrates the results. The time overhead of finding a satisfactory noise vector increases exponentially as $m$ increases. When $m$ reaches 11, it takes approximately 100 seconds on average to find a suitable noise vector (including 64 noises). For CCR-GAN, we adopt the same evaluation criteria as CCR-GAN, while $m$ varies from 21 to 25. Results are presented in Fig. 9(b). When $m$ is within the range of 21 to 23, the time overhead remains relatively consistent. When $m$ exceeds 24, the time overhead begins to increase significantly. The majority of boxes in Fig. 9 are positioned close to the maximum point, indicating that most results fall within the range near the maximum.

Table I provides more detailed time consumption for finding a satisfying noise vector and noise. All the means are higher than the medians, verifying that a majority of the results are close to the maximum. If the time consumption of the upper quartile is deemed acceptable, we consider the scheme to be feasible. For R-GAN with $m = 11$, the sender typically spends 165.125 and 2.580 seconds to find a satisfying noise vector and noise, which is deemed acceptable. In the case of CCR-GAN, the sender can accept the time overhead at $m = 24$. When $m = 25$, it may take 594.831 seconds and 9.134 seconds to find a noise vector and noise, which is unacceptable.

To summarize, R-GAN and CCR-GAN enable the embedding of 11 and 24 bits of covert data in each transaction amount, while maintaining an acceptable time overhead. We use these criteria for all subsequent experiments.

*2) Concealment:* **We evaluate the concealment using CTR.** The concealment experiments follow the same settings as [14], except for the dataset size. We use larger datasets to ensure reliable results. We generate 20,000, 40,000, 60,000, and 80,000 transaction amounts using R-GAN and CCR-GAN, respectively. These amounts, along with an equal number of normal transaction amounts, form the dataset. For each configuration, we perform 10 experiments and calculate the average result. Experiment results are summarized in Table II. For R-GAN, as the dataset size increases from 40,000 to 120,000, the evaluation criteria consistently increases in all cases. When the dataset size continues to increase to 160,000, the recognition probability of R-GAN remains similar to that at 120,000. Overall, R-GAN is recognized with an accuracy of 0.67. Similarly, CCR-GAN is recognized with an accuracy of 0.73. The result aligns with the intuitive understanding that as more bits are used to store covert data, fewer bits are available to match the features of normal amounts, resulting in a higher likelihood of detection. To conclude, CCR-GAN offers less concealment, while both schemes are challenging to detect.

TABLE II: Recognition results of CTR. The dataset is composed of positive and negative samples with a ratio of 1:1, and is divided into a training set and a test set with a ratio of 7:3. Each set consists of half the normal amount and the abnormal amount. The closer the result is to 0.5, the more concealment the scheme possesses.

| Scheme | Dataset Size | Accuracy | Precision | Recall | F1-score |
|---|---|---|---|---|---|
| R-GAN | 40,000 | 0.6229 | 0.6251 | 0.6229 | 0.6217 |
| | 80,000 | 0.6473 | 0.6522 | 0.6473 | 0.6445 |
| | 120,000 | 0.6600 | 0.6776 | 0.6600 | 0.6511 |
| | 160,000 | 0.6673 | 0.6762 | 0.6673 | 0.6631 |
| CCR-GAN | 40,000 | 0.7303 | 0.7303 | 0.7303 | 0.7302 |
| | 80,000 | 0.7281 | 0.7296 | 0.7281 | 0.7279 |
| | 120,000 | 0.7315 | 0.7358 | 0.7315 | 0.7303 |
| | 160,000 | 0.7326 | 0.7338 | 0.7326 | 0.7323 |

*3) Effectiveness of ClipSigmoid:* **We verify the effect of ClipSigmoid by plotting the loss function during training.** We compare the loss of CCR-GAN when using Sigmoid and ClipSigmoid as the activation function (with other parameters kept constant). Fig. 10 illustrates the impact of ClipSigmoid. At the 8th epoch, the model without ClipSigmoid (including both the generator and the discriminator) experiences a sudden spike and is difficult to converge again during subsequent training. Although ClipSigmoid only modifies a small portion of Sigmoid, it has a significant effect on model convergence.

### C. Comparison Experiments

We use CER to evaluate the capacity expansion capability of the proposed schemes against blockchain-based covert channels that do not utilize the transaction amount as the embedding field. For channels that do utilize the transaction amount for as the embedding field, we compare AC and concealment to demonstrate the superiority of our schemes.

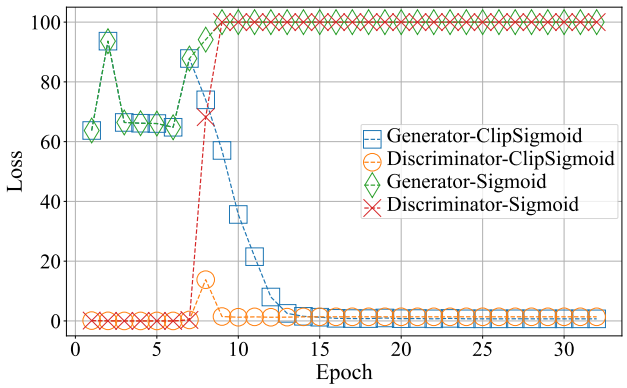



Fig. 10: The effect of ClipSigmoid.

*1) Non-amount as embedding field:* We consider that a Bitcoin transaction is a P2PKH[1] transaction with one input and two outputs by default, as this type of transaction accounts for the largest number [27]. AC of both the proposed schemes and baselines is calculated using this default setting. Table III presents CER of R-GAN and CCR-GAN compared to baselines. It can be observed that R-GAN increases the capacity of baselines by 4.30% to 91.67%, while CCR-GAN boosts it by 9.38% to 200.00%. Although amounts generated by CCR-GAN may slightly increase the identification risk, the huge capacity increase makes the risk worthwhile.

TABLE III: CER of R-GAN and CCR-GAN.

| Scheme | HC-CDE | DSA | Un-UTXO | DLchain |
|---|---|---|---|---|
| R-GAN | 91.67% | 4.30% | 6.88% | 8.59% |
| CCR-GAN | 200.00% | 9.38% | 15.00% | 18.75% |

*2) Amount as embedding field:* We also assume that Bitcoin transactions are P2PKH transactions with one input and two outputs. We perform the concealment experiments with the dataset of 16,000 settings. Comparison results in Table IV show that R-GAN has a smaller capacity compared to CCMBBT, while it exhibits the highest concealment. Besides, CCR-GAN has a larger capacity to CCMBBT, while its concealment surpasses that of CCMBBT. This is because CCMBBT simply encodes covert data as a number using the ASCII encoding, making their amounts easily distinguishable. R-GAN trades capacity for higher concealment and outperforms the baseline in concealment, while CCR-GAN outperforms the baseline in both capacity and concealment.

TABLE IV: Comparison with CCMBBT.

| Scheme | AC | Accuracy | Precision | Recall | F1-score |
|---|---|---|---|---|---|
| CCMBBT | 23 | 0.9095 | 0.9125 | 0.9095 | 0.9093 |
| R-GAN | 11 | 0.6673 | 0.6762 | 0.6673 | 0.6631 |
| CCR-GAN | 24 | 0.7326 | 0.7338 | 0.7326 | 0.7323 |

[1]Pay-to-Public-Key-Hash, a type of ScriptPubKey which locks Bitcoin to the hash of a public key.

## VI. Related Work

**Blockchain-based covert channels.** In addition to baselines, several works are relevant to blockchain-based covert channels. Partala [28] first builds a covert channel in the blockchain. They propose BLOCCE and demonstrate its security. BLOCCE stores 1-bit covert data into the least significant bit of the address. Gao *et al.* [29] and Zhang *et al.* [13] utilize OP_RETURN to encode covert data. Zhang *et al.* [30] construct a covert channel based on the parameters of Ethereum smart contracts. Luo *et al.* [26] represent bits 0 and 1 by the presence or absence of transactions between addresses, and also encode covert data into the transaction amount. Zhang *et al.* [31] employ the Ethereum Whisper protocol to establish a covert channe. Alsalami *et al.* [32] explore randomness in the blockchain to build covert channels. None of them consider generating required transaction fields, whereas our approaches specifically focus on generating these other fields.

**Transaction generation via AI.** Wang *et al.* [14] propose a PCTC model for generating transaction fields. They aim to generate indistinguishable transaction fields and provide a reference for creating covert transactions. Liu *et al.* [33] employ GAN to generate the Ethereum transaction fields and embed covert data. However, the receiver may not be unable to obtain decoded data consistent with the original covert data. Researchers also explore automating smart contract generation using AI [34]–[38], while none of them consider embedding covert data during the generation process. There is currently no research on embedding data while generating transaction fields via AI, whereas our approaches address this problem.

**AI-based text steganography.** The proposed schemes can also be considered as a form of text steganography. Existing research on text steganography focuses on linguistic steganography, where covert data is hidden within language semantics. Yang *et al.* [39] utilize variational auto-encoders to hide covert data into word selection. Their subsequent work enhances the concealment of generated sentences [40]. Li *et al.* [41] achieve steganographic long text generation by encoding covert data into entities and relationships within a knowledge graph. Zhou *et al.* [42] propose an adaptive embedding algorithm with a similarity function to implement linguistic steganography, ensuring that the embedded distribution remains consistent with the actual distribution. These methods do not apply to embedding covert data into purely numerical transaction fields.

## VII. Conclusion

This paper introduces GBSF, a generic framework for expanding the channel capacity of blockchain-based steganography. GBSF involves the sender generating indistinguishable required fields while embedding covert data. We instantiate the R-GAN scheme, which utilizes GAN with a reversible generator to generate the required fields and encodes covert data as input noise to the GAN generator. Additionally, we propose CCR-GAN as an enhancement to R-GAN, further improving its performance. Experimental results demonstrate that our proposed schemes outperform baselines.

## REFERENCES


[1] P. C. Mandal, I. Mukherjee, G. Paul, and B. Chatterji, "Digital image steganography: A literature survey," *Information Sciences*, 2022.

[2] Q. Liu, J. Yang, H. Jiang, J. Wu, T. Peng, T. Wang, and G. Wang, "When deep learning meets steganography: Protecting inference privacy in the dark," in *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*. IEEE, 2022, pp. 590–599.

[3] X. Zheng, Q. Dong, and A. Fu, "Wmdefense: Using watermark to defense byzantine attacks in federated learning," in *IEEE INFOCOM 2022-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2022, pp. 1–6.

[4] J. Qiang, S. Zhu, Y. Li, Y. Zhu, Y. Yuan, and X. Wu, "Natural language watermarking via paraphraser-based lexical substitution," *Artificial Intelligence*, vol. 317, p. 103859, 2023.

[5] M. B. Rosen, J. Parker, and A. J. Malozemoff, "Balboa: Bobbing and weaving around network censorship," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 3399–3413.

[6] Z. Zhou, X. Dong, R. Meng, M. Wang, H. Yan, K. Yu, and K.-K. R. Choo, "Generative steganography via auto-generation of semantic object contours," *IEEE Transactions on Information Forensics and Security*, 2023.

[7] Y. Xu, C. Mou, Y. Hu, J. Xie, and J. Zhang, "Robust invertible image steganography," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 7875–7884.

[8] T. Zhang, B. Li, Y. Zhu, T. Han, and Q. Wu, "Covert channels in blockchain and blockchain based covert communication: Overview, state-of-the-art, and future directions," *Computer Communications*, 2023.

[9] Y. Miao, Z. Liu, H. Li, K.-K. R. Choo, and R. H. Deng, "Privacy-preserving byzantine-robust federated learning via blockchain systems," *IEEE Transactions on Information Forensics and Security*, vol. 17, pp. 2848–2861, 2022.

[10] Z. Chen, L. Zhu, P. Jiang, C. Zhang, F. Gao, J. He, D. Xu, and Y. Zhang, "Blockchain meets covert communication: A survey," *IEEE Communications Surveys & Tutorials*, 2022.

[11] B. Du, D. He, M. Luo, C. Peng, and Q. Feng, "The applications of blockchain in the covert communication," *Wireless Communications and Mobile Computing*, vol. 2022, 2022.

[12] Z. Wang, L. Zhang, R. Guo, G. Wang, J. Qiu, S. Su, Y. Liu, G. Xu, and Z. Tian, "A covert channel over blockchain based on label tree without long waiting times," *Computer Networks*, p. 109843, 2023.

[13] C. Zhang, L. Zhu, C. Xu, Z. Zhang, and R. Lu, "Ebdl: Effective blockchain-based covert storage channel with dynamic labels," *Journal of Network and Computer Applications*, vol. 210, p. 103541, 2023.

[14] M. Wang, Z. Zhang, J. He, F. Gao, M. Li, S. Xu, and L. Zhu, "Practical blockchain-based steganographic communication via adversarial ai: A case study in bitcoin," *The Computer Journal*, vol. 65, no. 11, pp. 2926–2938, 2022.

[15] A. Creswell, T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta, and A. A. Bharath, "Generative adversarial networks: An overview," *IEEE signal processing magazine*, vol. 35, no. 1, pp. 53–65, 2018.

[16] K. Koptyra and M. R. Ogiela, "An extension of imagechain concept that allows multiple images per block," in *IEEE INFOCOM 2022-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2022, pp. 1–2.

[17] J. Zhang, X. Ji, W. Xu, Y.-C. Chen, Y. Tang, and G. Qu, "Magview: A distributed magnetic covert channel via video encoding and decoding," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 357–366.

[18] T. Yang, H. Wu, B. Yi, G. Feng, and X. Zhang, "Semantic-preserving linguistic steganography by pivot translation and semantic-aware bins coding," *IEEE Transactions on Dependable and Secure Computing*, 2023.

[19] J. Yang, Z. Yang, J. Zou, H. Tu, and Y. Huang, "Linguistic steganalysis toward social network," *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 859–871, 2022.

[20] J. Li, Y. Liu, W. Xu, and Z. Li, "Gasla: Enhancing the applicability of sign language translation," in *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*. IEEE, 2022, pp. 1249–1258.

[21] H. Cui, H. Bian, W. Zhang, and N. Yu, "Unseencode: Invisible on-screen barcode with image-based extraction," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 1315–1323.

[22] H. Cao, H. Yin, F. Gao, Z. Zhang, B. Khoussainov, S. Xu, and L. Zhu, "Chain-based covert data embedding schemes in blockchain," *IEEE Internet of Things Journal*, vol. 9, no. 16, pp. 14 699–14 707, 2020.

[23] A. Fionov, "Exploring covert channels in bitcoin transactions," in *2019 International Multi-Conference on Engineering, Computer and Information Sciences (SIBIRCON)*. IEEE, 2019, pp. 0059–0064.

[24] M. Gregoriadis, R. Muth, and M. Florian, "Analysis of arbitrary content on blockchain-based systems using bigquery," in *Companion Proceedings of the Web Conference 2022*, 2022, pp. 478–487.

[25] J. Tian, G. Gou, C. Liu, Y. Chen, G. Xiong, and Z. Li, "Dlchain: A covert channel over blockchain based on dynamic labels," in *Information and Communications Security: 21st International Conference, ICICS 2019, Beijing, China, December 15–17, 2019, Revised Selected Papers 21*. Springer, 2020, pp. 814–830.

[26] X. Luo, P. Zhang, M. Zhang, H. Li, and Q. Cheng, "A novel covert communication method based on bitcoin transaction," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 4, pp. 2830–2839, 2021.

[27] Bitcoin protocol layer statistics. Accessed: June 1, 2023. [Online]. Available: https://transactionfee.info/

[28] J. Partala, "Provably secure covert communication on blockchain," *Cryptography*, vol. 2, no. 3, p. 18, 2018.

[29] F. Gao, L. Zhu, K. Gai, C. Zhang, and S. Liu, "Achieving a covert channel over an open blockchain network," *IEEE Network*, vol. 34, no. 2, pp. 6–13, 2020.

[30] L. Zhang, Z. Zhang, W. Wang, Z. Jin, Y. Su, and H. Chen, "Research on a covert communication model realized by using smart contracts in blockchain environment," *IEEE Systems Journal*, vol. 16, no. 2, pp. 2822–2833, 2021.

[31] L. Zhang, Z. Zhang, Z. Jin, Y. Su, and Z. Wang, "An approach of covert communication based on the ethereum whisper protocol in blockchain," *International Journal of Intelligent Systems*, vol. 36, no. 2, pp. 962–996, 2021.

[32] N. Alsalami and B. Zhang, "Uncontrolled randomness in blockchains: Covert bulletin board for illicit activity," in *2020 IEEE/ACM 28th International Symposium on Quality of Service (IWQoS)*. IEEE, 2020, pp. 1–10.

[33] J. Liu, "Research on information hiding methods based on blockchain technology," Master's thesis, Nanjing University of Information Science and Technology, 2020.

[34] W. Shao, Z. Wang, X. Wang, K. Qiu, C. Jia, and C. Jiang, "Lsc: Online auto-update smart contracts for fortifying blockchain-based log systems," *Information Sciences*, vol. 512, pp. 506–517, 2020.

[35] V. Dwivedi and A. Norta, "Auto-generation of smart contracts from a domain-specific xml-based language," in *Intelligent Data Engineering and Analytics: Proceedings of the 9th International Conference on Frontiers in Intelligent Computing: Theory and Applications (FICTA 2021)*. Springer, 2022, pp. 549–564.

[36] A. Mavridou and A. Laszka, "Designing secure ethereum smart contracts: A finite state machine based approach," in *Financial Cryptography and Data Security: 22nd International Conference, FC 2018, Nieuwpoort, Curaçao, February 26–March 2, 2018, Revised Selected Papers 22*. Springer, 2018, pp. 523–540.

[37] H. Hu, Q. Bai, and Y. Xu, "Scsguard: Deep scam detection for ethereum smart contracts," in *IEEE INFOCOM 2022-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2022, pp. 1–6.

[38] J. He, J. Liu, Z. Zhang, Y. Chen, Y. Liu, B. Khoussainov, and L. Zhu, "Msdc: Exploiting multi-state power consumption in non-intrusive load monitoring based on a dual-cnn model," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, no. 4, pp. 5078–5086, 2023.

[39] Z. Yang, S. Zhang, Y. Hu, Z. Hu, and Y. Huang, "Vae-stega: Linguistic steganography based on variational auto-encoder," *IEEE Transactions on Information Forensics and Security*, vol. 16, 2021.

[40] Z. Yang, L. Xiang, S. Zhang, X. Sun, and Y. Huang, "Linguistic generative steganography with enhanced cognitive-imperceptibility," *IEEE Signal Processing Letters*, vol. 28, pp. 409–413, 2021.

[41] Y. Li, J. Zhang, Z. Yang, and R. Zhang, "Topic-aware neural linguistic steganography based on knowledge graphs," *ACM/IMS Transactions on Data Science*, vol. 2, no. 2, pp. 1–13, 2021.

[42] X. Zhou, W. Peng, B. Yang, J. Wen, Y. Xue, and P. Zhong, "Linguistic steganography based on adaptive probability distribution," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 5, pp. 2982–2997, 2021.