The ISC Int'l Journal of Information Security
ISeCure

October 2025, Volume 17, Number 3 (pp. 1–11)
http://www.isecure-journal.org

Presented at the ISCISC'2025 in Tehran, Iran.

Enhancing Kleptographic Backdoors in Hash-Based Deterministic Random Bit Generators **

Sepehr Jafari [1], and Raziyeh Salarifard [1,*]
[1] *Faculty of Computer Science and Engineering, Shahid Beheshti University, Tehran, Iran.*

ARTICLE INFO.

Keywords:
kleptographic backdoor, DRBG, Random number, NIST, ANN

Type:

doi:

ABSTRACT

Deterministic Random Bit Generators (DRBGs) are essential for cryptographic security but remain vulnerable to covert kleptographic attacks that implant backdoors to leak sensitive information. Despite being known for two decades, as demonstrated by incidents such as the Snowden revelations and Dual-EC, these attacks persist in modern protocols, including TLS and post-quantum systems. This paper introduces a novel kleptographic backdoor for hash-based DRBGs, utilising a dual-phase design: secret information is split across two complementary phases, each requiring the other for recovery. This design significantly increases the overall complexity compared with conventional methods. To enhance indistinguishability, we integrate randomness derived from the discrete logarithm problem, ensuring statistical conformity. By leveraging ElGamal encryption to ensure compatibility with our approach, we develop a highly covert backdoor. Rigorous validation via the NIST Statistical Test Suite (STS) and neural network-based anomaly detection confirms the backdoor passes all NIST tests while evading machine learning detection, maintaining statistical integrity and structural consistency.

1 Introduction

Deterministic Random Bit Generators (DRBGs) are critical components in cryptographic systems. They generate the pseudo-random numbers necessary for secure key generation, encryption, and digital signatures. The security of these systems relies heavily on the unpredictability of DRBG outputs. However, DRBGs, especially in black-box implementations, are vulnerable to attacks that exploit covert mechanisms to leak sensitive information.

A notable class of these attacks involves Secretly Embedded Trapdoors with Universal Protection (SETUP) and the broader concept of kleptography. In the early 1990s, researchers Young and Yung introduced a new category of attacks, termed kleptographic attacks, which focus on embedding backdoors within public-key cryptographic implementations, particularly in black-box environments [1]. In such black-box scenarios, a system designer may insert a hidden backdoor that compromises the system's security. Such backdoors compromise security, po-

* Corresponding author.
**The ISCISC'2025 program committee effort is highly acknowledged for reviewing this paper.
Email addresses: sep.jafari@mail.sbu.ac.ir, r_salarifard@sbu.ac.ir
ISSN: 2008-2045

tentially affecting the device's user and any entities interacting with the system.

This method enables attackers to embed hidden backdoors in cryptographic algorithms. Such backdoors can secretly leak sensitive information, such as the seed of a DRBG, while maintaining functional and statistical integrity, making them extremely difficult to detect using conventional testing methods. Although kleptography was introduced over two decades ago, striking recent examples—such as the Snowden revelations and the controversy surrounding Dual-EC [2]—have reawakened the security community to the seriousness of these issues [3].

This work introduces an innovative method for embedding kleptographic backdoors into Deterministic Random Bit Generators (DRBGs), building on foundational concepts in kleptography and addressing critical challenges in detecting such vulnerabilities. The key contributions of this paper can be summarised as follows:

- **Novel Backdoor Design with Complementary Infected Phase:** We propose a backdoor mechanism that splits the DRBG output into two distinct compromised phases, each produced through different generation methods. Unlike conventional approaches that alternate between valid and infected outputs, our method ensures that both infected phases are required to recover the secret information, thereby increasing the complexity and subtlety of the backdoor.
- **Integration of Discrete Logarithm-Based Randomness:** To enhance the backdoor's indistinguishability, we leverage the discrete logarithm problem to generate outputs in one of the infected phases. This approach emulates the behaviour of a secure random number generator while embedding the backdoor, ensuring compliance with the statistical properties expected from true cryptographic randomness.
- **Selecting the ElGamal Schema:** We select the ElGamal encryption scheme as the foundation for embedding the backdoor, given its compatibility with the discrete logarithm problem. This design choice supports our goal of developing a highly covert yet robust kleptographic backdoor that stays undetectable when tested with standard cryptographic analysis methods.
- **Validation Through Statistical and Machine Learning Analyses:** We evaluate the proposed backdoor by analysing the DRBG outputs using the NIST Statistical Test Suite (STS) and a neural network-based anomaly detection framework. The backdoor random numbers successfully pass all NIST STS tests. Additionally, our neural network-based anomaly detection model fails to identify any discernible anomalies in the outputs. These results highlight the covert nature of the proposed backdoor, which maintains both statistical integrity and structural consistency, making it exceptionally challenging to detect using conventional statistical tests or machine learning techniques.

Our findings highlight the increased sophistication and undetectability of such attacks in modern cryptographic systems, emphasising the urgent need for innovative detection mechanisms. The dual-phase design challenges current testing tools and provides a framework for developing future backdoor detection techniques in black-box systems.

2 Related Work

Recent research has applied the kleptography framework to the TLS protocol, a fundamental component of Internet security. In one study, researchers proposed a practical asymmetric backdoor for TLS versions 1.2 and 1.3. They successfully demonstrated this as a proof of concept within the OpenSSL library, confirming the viability of kleptographic attacks on modern TLS implementations [4].

Another study investigated embedding kleptographic backdoors in post-quantum cryptography (PQC), specifically targeting Kyber, a leading candidate for PQC standardisation. Researchers demonstrated practical methods for inserting backdoors into Kyber, utilising classical techniques like ECDH and post-quantum schemes such as Classic McEliece. This attack was validated at the protocol level, particularly in TLS 1.3, marking the first practical demonstration of such a kleptographic backdoor in a post-quantum cryptographic setting [5].

In post-quantum cryptographic schemes, another study examined rejection in the Fujisaki-Okamoto (FO) transform, a critical element in protocols such as CRYSTALS-Kyber. The research demonstrated how an attacker could compromise user code to weaken the security of the FO transform with implicit rejection while remaining covert and undetectable [6].

According to recent studies and given the practical applications of kleptography in post-quantum cryptographic algorithms and network security protocols, investigating current methods for identifying backdoors in black-box systems is increasingly important.

The NIST Statistical Test Suite (STS) is a widely recognised tool for performing stochastic analysis, commonly used to certify and validate random number generators. Since the NIST STS focuses solely on

analysing the outputs of a random number generator, it is especially suitable for black-box systems where users do not have access to the generator's internal states. However, if a backdoor is embedded within the generator in a way that preserves the expected statistical properties, the NIST STS may fail to detect such covert modifications. For instance, some DRBGs that exhibit strong randomness may successfully pass most statistical tests, yet still contain weak and inherent correlations that go undetected [7]. Therefore, it is essential to further investigate security analysis techniques and tools for PRNGs.

Recently, Artificial Neural Networks (ANNs) have gained significant attention for their strong ability to recognise patterns and uncover complex structures within large datasets [8][9]. This capability stems from fundamental principles, such as the universal approximation theorem, which asserts that multilayer feed-forward neural networks, even with a single hidden layer, can approximate any continuous function to arbitrary precision, provided the network possesses sufficient complexity [10].

Given this, researchers have increasingly turned to ANNs as powerful tools for analysing and distinguishing random sequences generated by pseudo-random algorithms and quantum processes [11][12][13][14]. Tuning various ANN parameters, such as the number of layers, nodes, and activation functions, enables the execution of highly complex computations. Consequently, ANNs offer a promising approach for testing the randomness of sequences, including verifying the unpredictability required for secure random number generation.

In this context, unpredictability is fundamental; it implies that the prior probability of a random sequence must equal its posterior probability. For a sequence to maintain its randomness and security, any deviation from this balance signals a potential vulnerability. Thus, applying ANN to verify the randomness of sequences, especially those generated by quantum processes, presents a new frontier in ensuring the security and reliability of random number generation.

3 Background

3.1 Deterministic Random Bit Generator (DRBG)

Random bit generators (RBGs) are integral to cryptographic systems, serving as the foundation for generating secure keys, initialisation vectors, and other critical parameters. The quality of randomness produced by an RBG directly impacts the security of the consuming application. NIST SP 800-90Ar-1 [15] defines two primary categories of RBGs based on their generation strategies:

(1) Non-Deterministic Random Bit Generators (NRBGs): These rely on unpredictable physical processes to produce bits, ensuring true randomness.
(2) Deterministic Random Bit Generators (DRBGs): These use a deterministic algorithm to generate pseudo-random bits from an initial seed value.

DRBGs are particularly significant in scenarios where physical randomness sources may not be available or practical. A DRBG is initialised using a seed derived from a source of entropy. Upon instantiation, the DRBG uses an algorithm to expand the seed into a pseudo-random bit sequence. For the DRBG to be secure:

- The seed must contain sufficient entropy to ensure randomness.
- The seed must remain secret.
- The DRBG algorithm must be well-designed and resistant to attacks.

NIST highlights that the security of implementation depends on both the DRBG mechanism and the quality of its entropy source. The generated pseudo-random bits should maintain unpredictability up to the security strength of the DRBG, ensuring suitability for cryptographic applications.

3.1.1 Hash-Based Deterministic Random Bit Generators (Hash DRBGs)

Hash DRBGs are a specific class of DRBGs defined within NIST SP 800-90Ar-1. They leverage cryptographic hash functions to produce pseudo-random bits. The robustness of Hash DRBG stems from the security properties of the underlying hash function, including:

- **Collision Resistance:** Ensures that it is computationally infeasible to find two distinct inputs that produce the same hash output.
- **Preimage Resistance:** Prevents attackers from reconstructing the original input from the hash output.
- **Second-Preimage Resistance:** Ensures that finding another input that produces the same hash output as a given input is computationally infeasible.

The core mechanism of a Hash DRBG involves initialising the generator with a seed derived from an entropy source. The hash function is then applied iteratively to expand the seed into a pseudo-random output. This process is designed to be efficient while retaining the security guarantees provided by the

Table 1. Definitions for Hash-Based DRBG Mechanisms

	SHA-1	SHA-224 / SHA-512/224	SHA-256 / SHA-512/256	SHA-384	SHA-512
Output Block Length	160	224	256	384	512
Seed length for Hash_DRBG	440	440	440	888	888

hash function. Table 1 summarises the output block length and seed length requirements for Hash DRBG when different cryptographic hash functions are used. Consequently, the maximum seed length of 888 bits establishes the upper bound on the data that a backdoor must leak.

Hash DRBGs are widely used due to their simplicity, efficiency, and reliance on well-established cryptographic primitives. However, their deterministic nature also presents a potential vulnerability. Any compromise of the seed or the introduction of a backdoor into the mechanism could undermine the entire system's security. Consequently, rigorous testing and analysis, such as those provided by the NIST Statistical Test Suite (STS), are essential to ensure the integrity and unpredictability of the generated outputs.

3.1.2 Linear congruential generator (LCG)

To evaluate the predictive capabilities of our ANN model, a Linear Congruential Generator (LCG) [16] is employed as a representative deterministic mechanism. LCGs are widely used in software platforms because they generate sequences with known and potentially long periods, provided appropriate parameters are selected. The algorithm of LCG is described by the recurrence relation:

$$X_{n+1} = (aX_n + c) \mod M, \tag{1}$$

Where X denotes the sequence of pseudo-random numbers, and M, a, and c are constants representing the modulus, multiplier, and increments, respectively. With suitable parameter choices, the generator can achieve a period equal to M for any initial seed value. The conditions for generating pseudo-random numbers with the full period are: (1) M and c must be relatively prime. (2) a-1 must be divisible by all prime factors of M. (3) If M is divisible by 4, then a-1 must also be divisible by 4. We used Python to generate random numbers, with the parameters listed in Table 2.

Table 2. Parameters employed by LCG to generate random sequences.

Parameter	Value
M	$2^{24}, 2^{26}, 2^{28}$
a	22695477
c	1

3.2 Kleptography: Exploiting Cryptosystems Covertly

Subliminal channels pose a significant concern in cryptosystems. Attackers can exploit these unintended pathways to leak sensitive information. These channels enable the transmission of concealed data, seed, or private information to unauthorised parties. However, because subliminal channels expose the hidden data universally, they can sometimes be detected under specific conditions.

A more sophisticated class of attacks, known as Secretly Embedded Trapdoors with Universal Protection (SETUP), takes this concept further. SETUP mechanisms, introduced in the mid-1990s [17], allow attackers to embed unnoticeable modifications into cryptosystems operating in black-box environments. These mechanisms are particularly insidious as they resist reverse engineering and maintain the cryptographic system's apparent integrity. SETUP mechanisms integrate subliminal channels with public key cryptography, enabling exclusive and covert access for the attacker.

Kleptography, a term coined to describe the study of securely and subliminally stealing information, builds on the SETUP framework. In kleptographic attacks, an attacker leverages a cryptosystem to extract sensitive information—such as private keys or seeds— while maintaining statistical and functional indistinguishability from legitimate operations. This method enables the attacker to operate covertly without arousing suspicion from users or security audits. We begin with a formal description of an asymmetric backdoor adopted from [1].

3.3 Discrete Logarithm and ElGamal Scheme

The discrete logarithm problem is defined as finding x in the equation $g^x \mod p = y$, where g is a generator of a cyclic group Z_p^*, p is a prime number, and y is the result of modular exponentiation. According to the work of Gennaro [18], the discrete logarithm can be leveraged to construct efficient pseudo-random number generators (PRNGs). This approach assumes that solving the discrete logarithm problem modulo a prime p is computationally infeasible.

In our study, the ElGamal scheme's inherent pseudo-randomness, derived from the discrete logarithm problem, is utilised to construct a secure

pseudo-random space [19].

Here is the ElGamal encryption and decryption process:

Encryption Algorithm

1. Public parameters (g, q, y) are determined.
2. A random number r is chosen.
3. The cipher pair (C_1, C_2) is calculated using the following formulas:

$$C_1 = g^r \mod q \tag{2}$$
$$C_2 = p \cdot y^r \mod q \tag{3}$$

Where p is the plaintext message to be encrypted.

Decryption Algorithm

1. Using the private key x, calculate $C_1^x \mod q$.
2. Recover the plaintext message p by multiplying C_2 with the inverse of the result from the previous step:

$$p = C_2 \cdot (C_1^x)^{-1} \mod q \tag{4}$$

3.4 Artificial Neural Network (ANN)

ANNs are computational models inspired by biological neural networks. They consist of layers of interconnected nodes (neurons) that process data in a hierarchical manner.

The Universal Approximation Theorem (UAT) provides a theoretical foundation for the capabilities of ANNs [10]. It states that a feed-forward neural network with at least one hidden layer, equipped with a non-linear activation function, can approximate any continuous function on a compact subset of $\mathbb{R}^n$ to an arbitrary degree of accuracy, given sufficient neurons in the hidden layer. This property highlights the ability of the ANNs to approximate complex functions, making it well-suited for tasks involving intricate dependencies and non-linear relationships.

4 Proposed Work

4.1 Target side

The Hash DRBG mechanism for generating random numbers follows a structural process. During the instantiation phase, an initial seed is generated based on the provided entropy source. When the user requires a new random number, the *generate*() function is called. This function employs a secondary function to generate the random output using the predetermined hash function specified during instantiation. After generating the random number, the *generate*() function updates the seed, tracks the generation count via an internal counter, and returns the output to the user. Refer to Figure 1 for a detailed illustration of this mechanism.

To embed the backdoor, we focus on the *hashgen*() function, which is responsible for generating random bits. This function utilises the internal seed to derive random bits through a hashing process, making it an effective target for covert access to the seed.

The backdoor leverages the ElGamal encryption scheme. Initially, the public and private parameters associated with ElGamal are generated. Public parameters are embedded within the target device, enabling the encryption of data designated for leakage. The private parameters are retained on the attacker's side to facilitate the recovery of the leaked information.

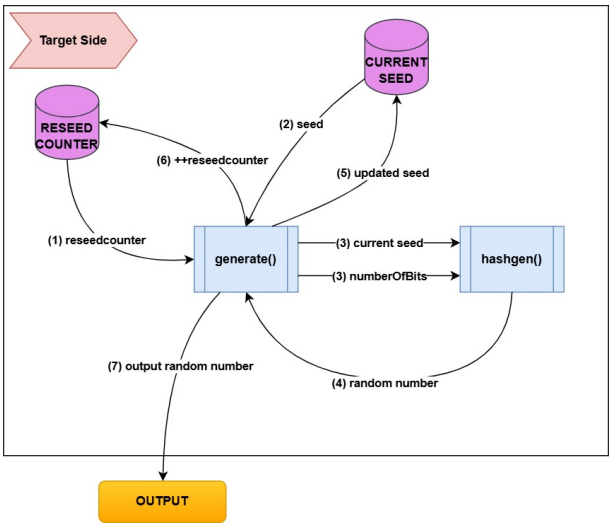

Figure 1. Random number generation process using an uninfected HASH DRBG on the target side.

Within the *hashgen*() function, before the random number is returned to the *generate*() function, it is passed to the malicious function along with the current seed and the length of the random number. This malicious function encrypts the current seed using the ElGamal scheme, storing the result in two distinct components, C1 and C2. Subsequently, with each invocation of the *generate*() function, portions of these encrypted components are embedded into specific bytes of the generated random number. For a detailed depiction of the modified Hash DRBG mechanism, refer to Figure 2.

To enable the backdoor mechanism, specific public parameters must be embedded within the target device:

- g: A generator for $\mathbb{Z}_p^*$.
- q: A large prime number.

- y: A public key, computed as $y = g^x \mod q$, where x is the corresponding private key.

These parameters serve as the foundation for the encryption schema employed in the malicious function. The *malFunc*() function uses these public parameters during two distinct phases to generate two encrypted components, C1 and C2 (Equation 2 and Equation 3), which systematically leak the current seed.

Here, r is a random number generated by the hashgen() function, and p represents the current seed. For all computations, we ensure the use of the seed's positive value. This approach guarantees consistent behaviour during encryption and embedding, preventing errors associated with negative values.

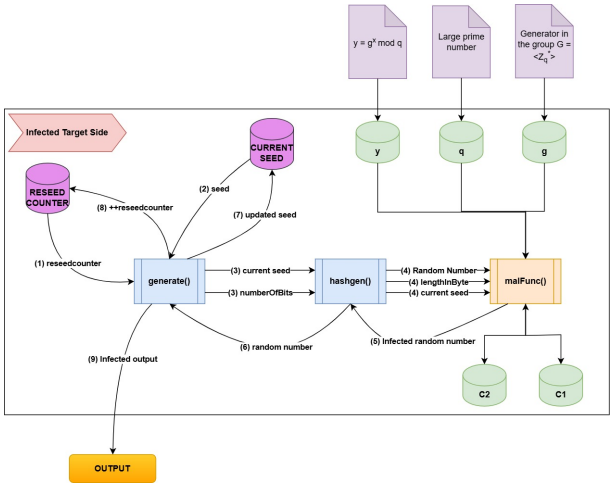

Figure 2. Random number generation process in the infected HASH DRBG, demonstrating the usage of the malicious function on the target side.

The Hash DRBG uses an internal counter to track the count of generated outputs (or seeds). This counter serves two primary functions:

(1) To determine whether the current seed is the first seed, in which case the components C_1 and C_2 must be generated and stored.
(2) To distinguish between odd and even iterations during the random number generation process.

In the backdoor design, pseudo-random values from the encrypted component C1 are embedded into specific positions of the random number during odd iterations. Specifically, the values are inserted into the byte positions that are multiples of 7 within the random number generated by the *hashgen*() function. Similarly, in even iterations, values from component C2 are embedded at byte positions divisible by 7.

To implement this mechanism, the following internal parameters are defined:

- A `table1counter` to track the number of bytes leaked from component C_1.
- A `table2counter` to track the number of bytes leaked from component C_2.
- An update flag to indicate whether all values in C_1 and C_2 have been fully leaked.

Given the maximum seed length is 111 bytes (as derived from Table 1), the backdoor mechanism ensures that, once the counters confirm all 111 bytes have been leaked, the encrypted components C1 and C2 are updated using the current seed. This process guarantees continuous, systematic leakage of the seed while maintaining the statistical properties of the Hash DRBG's output. For the detailed description, refer to Figure 3.

4.2 Attacker side

On the attacker's side, seed recovery consists of three distinct phases: data collection, data extraction, and decryption. These steps are executed systematically as follows:

(1) **Data Collection:** The attacker collects a sufficient number of random numbers generated by the infected Hash DRBG. The number of required random numbers depends on the length of the random numbers requested by the user. The backdoor algorithm embeds values from C_1 and C_2 into specific byte positions of base random numbers. If a single random number is sufficiently long, all necessary data can be extracted within two iterations. Conversely, for shorter outputs lacking sufficient embedding space, the attacker must collect additional random numbers to fully recover the data.
(2) **Data Extraction:** Once the required set of random numbers is gathered, the attacker uses a *secretRecovery()* function to extract the embedded values.
(3) **Decryption and Seed Recovery:** After extracting all the necessary values from components C_1 and C_2, the attacker proceeds to decrypt the data using their private key. The decryption process leverages the ElGamal encryption scheme, where the attacker uses their private key x and the public parameters q and g to recover the original seed.

The decryption is performed in the *seedRecovery()* function, which reconstructs the seed. However, since the encryption process on the target side always uses the positive value of the seed, the attacker cannot be certain whether the original seed was positive or negative before encryption. Therefore, after decryption, the attacker must compute both the positive and negative values of the recovered seed to ensure a correct sign and the robustness of the

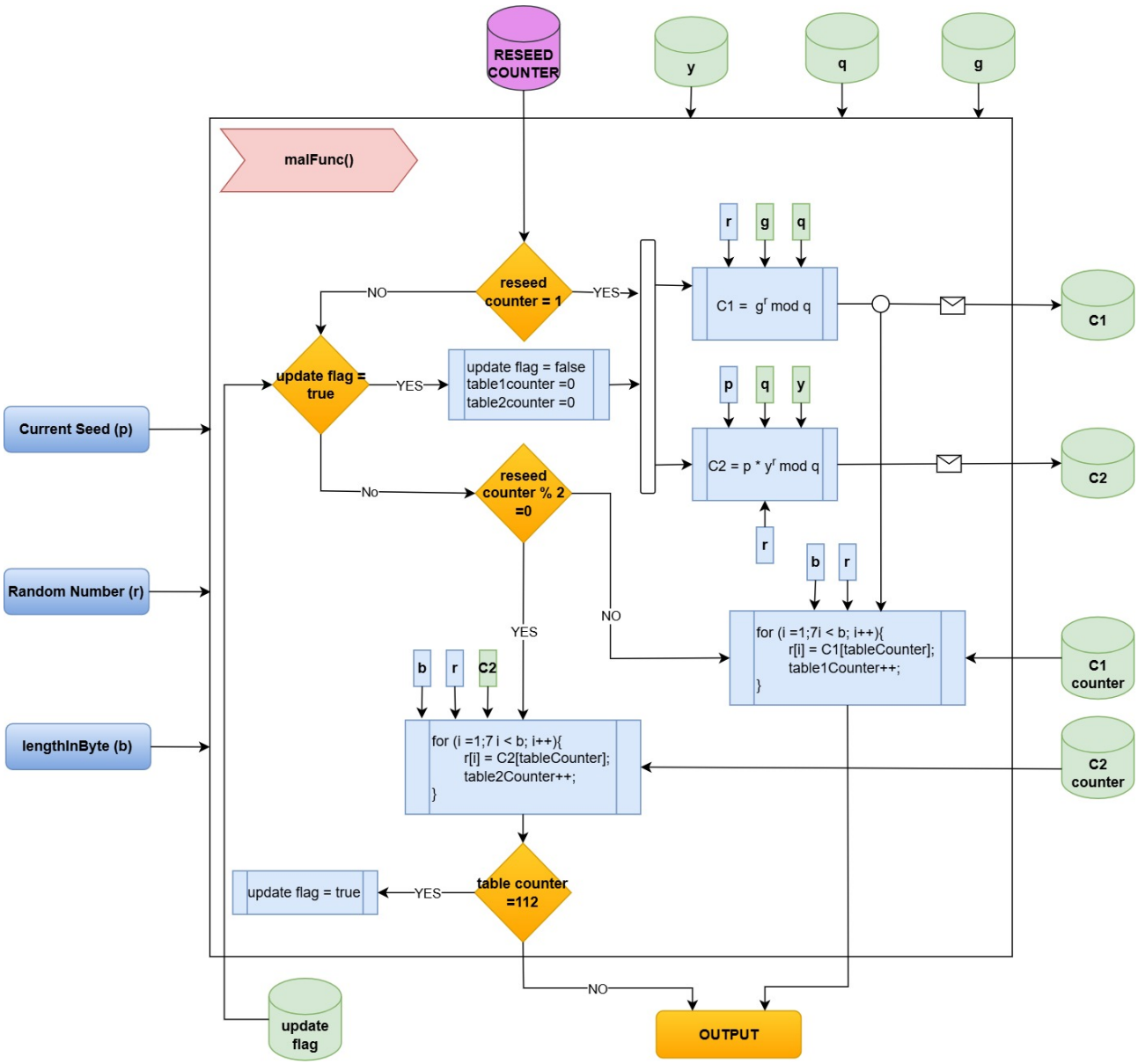

Figure 3. Structure of the malicious function, illustrating the use of the ElGamal cipher scheme and the process for managing information insertion into the output.

backdoor mechanism.

This systematic approach enables the attacker to recover the original seed while remaining undetected, as the leaked data is embedded to preserve the statistical appearance of randomness.

4.3 ANN model

In this study, we employed a fully connected feed-forward artificial neural network (ANN) to predict the seventh bit of a random number based on its preceding six bits. The 6-64-32-2 configuration was selected after iterative experimentation to ensure the model could effectively approximate the relationships in the input data without overfitting. The architecture is shown in Figure 5.

Let $x \in \mathbb{R}^6$ denote the input feature vector. The model computes the output $y \in \mathbb{R}^2$ through the following steps:

(1) $h_1 = \text{ReLU}(W_1 x + b_1)$, where $W_1 \in \mathbb{R}^{64\times 6}$ and $b_1 \in \mathbb{R}^{64}$ are the weight matrix and bias vector of the first hidden layer.
(2) $h_2 = \text{ReLU}(W_2 h_1 + b_2)$, where $W_2 \in \mathbb{R}^{32\times 64}$ and $b_2 \in \mathbb{R}^{32}$ represent the parameters of the second hidden layer.
(3) $y = \text{Sigmoid}(W_3 h_2 + b_3)$, where $W_3 \in \mathbb{R}^{2\times 32}$ correspond to the parameters of the output layer.

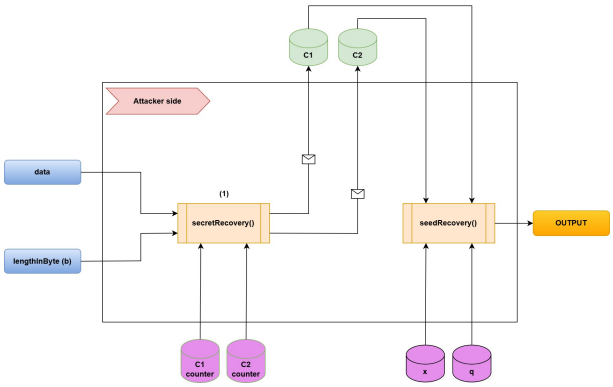

Figure 4. Process of data collection and seed recovery using a fixed private key on the attacker side.

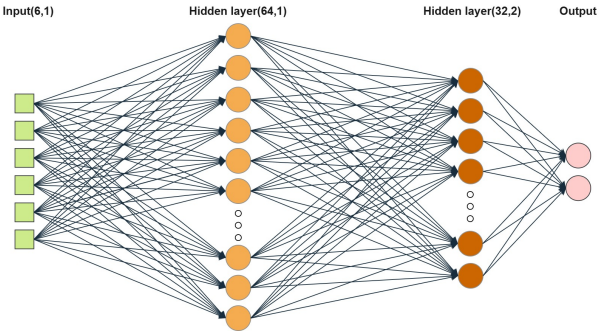

Figure 5. Architecture of the artificial neural network (ANN). The network consists of an input layer with 6 neurons, two hidden layers with 64 and 32 neurons, respectively, and an output layer with 2 neurons.

The network was trained on binary sequences using Cross-Entropy Loss for classification and the Adam optimiser with a learning rate of $\alpha = 0.001$. Training was performed for 40 epochs, ensuring sufficient iterations for convergence. We intend to predict the seventh bit from its preceding six consecutive bits.

5 Data Collection and Preprocessing

We generated random numbers using the Linear Congruential Generator (LCG), the Hash DRBG, and the infected Hash DRBG to train and evaluate the ANN model. Each method produced a sequence of decimal random numbers, which were subsequently binarised by applying a threshold value of 5. This threshold converts each decimal value to binary format, producing a continuous bitstream.

The training and test datasets were constructed by segmenting the binary stream into non-overlapping groups of seven consecutive binary values. Each group comprises six input bits and a seventh target bit serving as the label. This configuration is illustrated in Figure 6. Our task is to learn hidden correlations among the random numbers generated by RNGs and predict the next bit based on observed random bits in an input sequence.

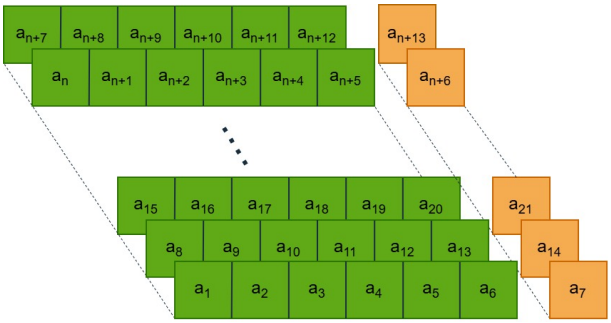

Figure 6. The method for obtaining the training and test instances. Each instance consists of seven random numbers intercepted from the original sequence without overlap. Six numbers are used as network inputs, while the seventh serves as the label.

The training process utilised a dataset comprising 4,000,000 instances. For evaluation, the dataset was partitioned into ten subsets, each containing 1,000,000 instances. Each subset was divided into ten tests, each containing 100,000 instances.

6 System Evaluation

The success rate for predicting a binary sequence in a randomness test is expected to hover around 50% for truly random datasets, since each bit has an equal probability of being 0 or 1. However, due to pseudo-randomness, the success rate will not always be exactly 50%. There will be natural fluctuations around this value, depending on the number of predictions (n).

The variance is:

$$\begin{aligned} \mathrm{Var}(Y) &= np(1-p) \\ &= \frac{\mathrm{Var}(Y)}{n^2} \\ &= \frac{np(1-p)}{n^2} \\ &= \frac{p(1-p)}{n} \\ &= \frac{1}{4n} \end{aligned} \tag{5}$$

The 3σ and 5σ boundaries define ranges of values around the expected mean (50%) where the observed success rate reflects random natural fluctuations. These boundaries are derived using the standard deviation (σ).

The standard deviation is:

$$\sigma = \sqrt{\mathrm{Var}(Y)} = \frac{1}{2\sqrt{n}} \tag{6}$$

Exceeding the $0.5 + 3\sigma$ threshold—a rare event—implies 99.73% confidence that the ANN learned from the training set. Furthermore, success rates surpassing the $0.5 + 5\sigma$ bound strongly suggest

an extremely high probability that the observed deviation stems not from random chance, but from the model detecting latent patterns within the data.

7 Experimental Result

The randomness of the evaluated PRNGs was first assessed using the NIST Statistical Test Suite (STS), which applies 15 statistical tests to detect non-randomness in binary sequences. Each test outputs a P-value, where P-values exceeding the 0.01 threshold indicate no statistically significant deviation from randomness. Table 3 presents the results for the Linear Congruential Generator (LCG) and Hash-based Deterministic Random Bit Generator (Hash DRBG).

Table 3. Results of NIST STS on the datasets at different periods of LCG, and for infected and uninfected Hash DRBG.

	LCG[14]			Hash DRBG	
Statistical Test	2^{24}	2^{26}	2^{28}	Infected	UnInfected
Frequency	Success	Success	Success	Success	Success
Block Frequency	Success	Success	Success	Success	Success
Cumulative Sums	Success	Success	Success	Success	Success
Runs	Success	Success	Success	Success	Success
Longest Run	Success	Success	Success	Success	Success
Rank	Success	Success	Success	Success	Success
FTT	Failure	Success	Success	Success	Success
Non-overlapping Template	Failure	Failure	Success	Success	Success
Overlapping Template	Success	Success	Success	Success	Success
Universal	Success	Success	Success	Success	Success
Approximate Entropy	Failure	Success	Success	Success	Success
Random Excursions	Success	Success	Success	Success	Success
Random Excursions Variant	Success	Success	Success	Success	Success
Serial	Failure	Success	Success	Success	Success
Linear Complexity	Success	Success	Success	Success	Success
Total successful tests	11/15	14/15	15/15	15/15	15/15

For the LCG, the number of passing tests improved with larger values of M. Deviations were detected at $M = 2^{24}$ and $M = 2^{26}$, but at $M = 2^{28}$, the generator passed all 15 tests. By contrast, both the uninfected and backdoored Hash DRBG consistently passed every test. These findings highlight a key limitation of NIST STS [20]. Although effective at detecting statistical weaknesses in poorly designed PRNGs, it cannot identify kleptographic backdoors in secure generators, since such mechanisms preserve statistical properties.

To further investigate subtle dependencies, we implemented an Artificial Neural Network (ANN) using PyTorch on Google Colab, utilising GPU/TPU acceleration. The ANN was evaluated on sequences from the LCG ($M = 2^{28}$) using 10 independent datasets, each containing 100,000 samples. Six consecutive bits served as input features, while the seventh functioned as the prediction target. As shown in Figure 7, prediction accuracy ranged from 0.505 to 0.513, consistently exceeding the statistical thresholds (0.505–0.508), thereby confirming the presence of exploitable correlations in the LCG output.

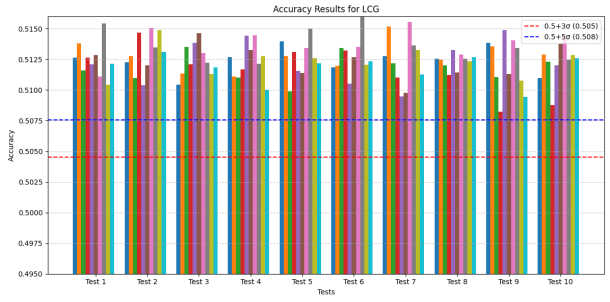

Figure 7. Test results for the Linear Congruential Generator (LCG). Each experiment consists of ten test subsets, each containing 100,000 cases, with each case comprising seven bits. The experiment is repeated ten times. The red and blue lines represent the 3σ and 5σ bounds, respectively. The prediction success rate for the LCG consistently exceeds the 5σ bound, indicating that the randomness generated by this algorithm is poor.

Figure 8 presents the ANN results for both infected and uninfected Hash DRBG outputs. In this case, prediction accuracies remained within random guessing boundaries, with no consistent exceedance beyond the $0.5 + 3\sigma$ limit. This result suggests that the kleptographic backdoor does not alter observable statistical properties, effectively concealing the seed-leakage mechanism from both NIST STS and ANN-based analysis.

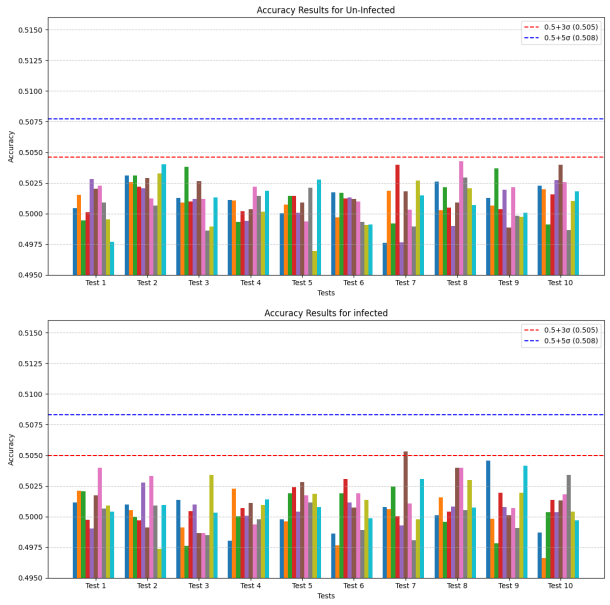

Figure 8. Testing results for infected and uninfected Hash DRBG outputs. The results indicate that the model fails to learn any meaningful correlation in the data, as it does not achieve the 3σ or 5σ thresholds.

8 Conclusion

In this paper, we have explored the growing threat of kleptographic attacks on Hash DRBGs, particularly in black-box cryptographic systems. We introduced a novel dual-phase backdoor design to demonstrate how attackers can embed covert, sophisticated backdoors. These backdoors successfully evade detection by conventional statistical tests and machine learning-based anomaly detection frameworks. Our approach leverages the discrete logarithm problem and the ElGamal encryption scheme to ensure the backdoor's indistinguishability and statistical integrity, making it exceptionally challenging to identify using existing tools.

The successful validation of our backdoor through the NIST STS and a neural network-based anomaly detection framework highlights the limitations of current testing methodologies in detecting such advanced threats. These findings underscore the urgent need for innovative detection mechanisms that can address the increasing sophistication of kleptographic attacks. Our work advances the understanding of kleptographic vulnerabilities and establishes a foundation for future techniques to identify and mitigate backdoors in cryptographic systems. As cryptographic systems continue to evolve, particularly with the advent of post-quantum cryptography, the risk of covert attacks is expected to escalate. This paper serves as a call to action for the cryptographic community to prioritise the development of robust, adaptive, and comprehensive detection tools. By doing so, we can better safeguard the integrity and security of cryptographic systems against the ever-present threat of kleptographic attacks.

References

[1] Adam Young and Moti Yung. Kleptography: Using cryptography against cryptography. In *Advances in Cryptology—EUROCRYPT'97: International Conference on the Theory and Application of Cryptographic Techniques Konstanz, Germany, May 11–15, 1997 Proceedings 16*, pages 62–74, 1997.

[2] Daniel J Bernstein, Tanja Lange, and Ruben Niederhagen. Dual ec: A standardized back door. pages 256–281. Springer, 2016.

[3] Phillip Rogaway. The moral character of cryptographic work. *Cryptology ePrint Archive*, 2015.

[4] Adam Janovsky, Jan Krhovjak, and Vashek Matyas. Bringing kleptography to real-world tls. In *Information Security Theory and Practice: 12th IFIP WG 11.2 International Conference, WISTP 2018, Brussels, Belgium, December 10–11, 2018, Revised Selected Papers 12*, pages 15–27, 2019.

[5] Prasanna Ravi, Shivam Bhasin, Anupam Chattopadhyay, Aikata Aikata, and Sujoy Sinha Roy. Backdooring post-quantum cryptography: Kleptographic attacks on lattice-based kems. In *Proceedings of the Great Lakes Symposium on VLSI 2024*, pages 216–221, 2024.

[6] Antoine Joux, Julian Loss, and Benedikt Wagner. Kleptographic attacks against implicit rejection. 2024.

[7] Darren Hurley-Smith and Julio Hernandez-Castro. Certifiably biased: An in-depth analysis of a common criteria eal4+ certified trng. *IEEE Transactions on Information Forensics and Security*, 13(4):1031–1041, 2017.

[8] Felix Brockherde, Leslie Vogt, Li Li, Mark E Tuckerman, Kieron Burke, and Klaus-Robert Müller. Bypassing the kohn-sham equations with machine learning. *Nature communications*, 8(1):872, 2017.

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[10] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[11] Yulong Feng and Lingyi Hao. Testing randomness using artificial neural network. *IEEE Access*, 8:163685–163693, 2020.

[12] Fenglei Fan and Ge Wang. Learning from pseudo-randomness with an artificial neural network–does god play pseudo-dice? *IEEE Access*, 6:22987–22992, 2018.

[13] Cai Li, Jianguo Zhang, Luxiao Sang, Lishuang Gong, Longsheng Wang, Anbang Wang, and Yuncai Wang. Deep learning-based security verification for a random number generator using white chaos. *Entropy*, 22(10), 2020.

[14] Nhan Duy Truong, Jing Yan Haw, Syed Muhamad Assad, Ping Koy Lam, and Omid Kavehei. Machine learning cryptanalysis of a quantum random number generator. *IEEE Transactions on Information Forensics and Security*, 14(2):403–414, 2019.

[15] Elaine B Barker, John Michael Kelsey, et al. *Recommendation for random number generation using deterministic random bit generators (revised)*. US Department of Commerce, Technology Administration, National Institute of Standards and Technology, Computer Security Division, Information Technology Laboratory, Washington, DC, USA, 2007.

[16] Joan Boyar. Inferring sequences produced by a linear congruential generator missing low-order bits. *Journal of Cryptology*, 1(3):177–184, 1989.

[17] Adam Young and Moti Yung. The dark side of "black-box" cryptography or: Should we trust capstone? In *Advances in Cryptology—CRYPTO'96: 16th Annual International Cryptology Conference Santa Barbara, California, USA August 18–22, 1996 Proceedings 16*, pages 89–103. Springer, 1996.

[18] Rosario Gennaro. An improved pseudo-random generator based on the discrete logarithm problem. *Journal of Cryptology*, 18:91–110, 2005.

[19] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, 31(4):469–472, 1985.

[20] Lawrence E Bassham, Andrew L Rukhin, Juan Soto, James R Nechvatal, Miles E Smid, Stefan D Leigh, M Levenson, M Vangel, Nathanael A Heckert, and D L Banks. A statistical test suite for random and pseudorandom number generators for cryptographic applications. 2010.

Sepehr Jafari graduated from Shahid Beheshti University with a degree in Computer Engineering. He has research and practical experience in cryptography, having worked on related projects at Shahid Beheshti University. In addition, he is skilled in web application penetration testing and is currently working as a penetration tester at Kashef Company. His professional interests include cryptography, cybersecurity, and secure system design.

Raziyeh Salarifard received the B.Sc. degree from the Sharif University of Technology, Tehran, Iran, in 2012, the M.Sc. and PhD degrees from the same university, in 2014 and 2019, respectively, all in computer engineering (hardware). She is now working as an Assistant Professor at the Faculty of Computer Science and Engineering, Shahid Beheshti University. Her research interests include hardware security, cryptographic engineering, and secure, efficient computing and architectures.