Anti-CSRF Token Using Linear Congruential Generator

MSc Research Project
MSc Cyber Security

Abraham Samson Nadar
Student ID: X21155089

School of Computing
National College of Ireland

Supervisor: Michael Prior

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Abraham Samson Nadar		
Student ID:	X21155089		
Programme:	MSc Cyber Security	**Year:**	September 2022
Module:	MSc Research Project		
Supervisor:	Michael Prior		
Submission Due Date:	14/08/2023		
Project Title:	Anti-CSRF Token Using Linear Congruential Generator		
Word Count:	7774	**Page Count**	20

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Abraham Samson Nadar
Date:	09/08/2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Anti-CSRF Token Using Linear Congruential Generator

Abraham Samson Nadar
X21155089

Abstract

This work describes a novel Anti-CSRF token generating method that uses the Enhanced Linear Congruential Generator (eLCG) in combination with HMAC_DRBG. This method improves the unpredictability and security of CSRF tokens, hence strengthening web application defences. The major goal is to evaluate the usefulness of eLCG & HMAC_DRBG based tokens in preventing CSRF attacks. To reach this purpose, the research design is methodically created, with the eLCG coupled with HMAC_DRBG to establish a robust foundation for creating unpredictable CSRF tokens. To validate the quality and reliability of the generated tokens, statistical tests are performed, indicating excellent quality, uniform distribution, lack of correlation, and absence of non-random patterns. Scatter plots depicts the generated CSRF numbers visually, emphasizing the uniqueness of each token and the absence of known trends. However, actual implementation considerations, such as adequate administration and secure storage of cryptographic keys and seed values, must be considered. The suggested technique will be integrated into leading cloud platforms such as Google Cloud Platform (GCP) and Microsoft Azure in the future, with the goal of strengthening identity protection and improving security across varied online settings. Adopting adaptive token generation algorithms led by real-time threat assessments has the potential to improve the robustness of CSRF token mechanisms even further. Finally, by developing a strong Anti-CSRF token generation system, this research greatly adds to online application security. The successful combination of eLCG and HMAC_DRBG demonstrates a formidable defence against CSRF assaults. As online applications expand, this research lays the groundwork for sophisticated security measures and future advancements, ultimately protecting digital interactions and user data from CSRF vulnerabilities.

1 INTRODUCTION

This study seeks to construct an Anti-CSRF Token using the Linear Congruential Generator (LCG) in order to address the issues raised in the publication [1]. The LCG creates repeated random numbers that are potentially predictable and can be used to attack a website's vulnerability. This vulnerability is a major motivation to address and a deciding element in topic selection. To address this issue, the seed value for each sequence can be modified manually, increasing the CSPRNG's robustness. The cryptosystem's security is based on seeding a CSPRNG algorithm with the most entropy achievable. By including a random seed value, the LCG becomes more unpredictable, resolving the issue. This solution's main contribution is determining the efficiency of LCG when paired with an additional algorithm for its further use.

CSRF attacks have been addressed in multiple study articles, with various mitigating approaches presented. One such approach is the Enhanced LCG (Linear Congruential

Generator), which combines the best features of both HMAC_DRBG and LCG to generate a safe and effective pseudorandom number generation procedure. The Enhanced LCG offers numerous benefits over typical LCGs or independent HMAC_DRBG methods, including security, efficiency, and computational efficiency.

Research Question: How effectively are anti-CSRF tokens created with a linear congruential generator to prevent CSRF in websites?

1.1 OBJECTIVES

1. Create and deploy an eLCG-based algorithm for issuing safe CSRF tokens.
2. Analyze the unpredictability & randomness of the CSRF tokens that are created.
3. Examine the CSRF token sequence's uniformity & independence.
4. Identify & investigate potential non-random patterns within the CSRF token sequence.

This study aims to demonstrate the effectiveness of the eLCG-based Anti-CSRF Token mechanism in preventing CSRF attacks by ensuring the tokens' unpredictability, randomness, uniformity, and independence while detecting and mitigating non-random patterns by conducting these tests and analysing their results. The usage of HMAC_DRBG-seeded eLCG improves the overall security of CSRF tokens and contributes to online application security robustness.

In order to respond to the study topic, the evaluation seeks to produce a series of Cross-Site Request Forgery (CSRF) numbers using the HMAC_DRBG (HMAC-based Deterministic Random Bit Generator) & eLCG. The HMAC_DRBG seeded eLCG, which generates an array of series & is used to generate the CSRF numbers. While the eLCG formula is derived from the LCG's formula, the eLCG will not be taking its own seed.

The eLCG is the best option for services needing reliable pseudorandom number generation, such as the creation of anti-CSRF tokens in web applications. It strikes an acceptable balance between cryptographic safety and efficiency. Using HMAC_DRBG to generate a cryptographically secure random seed and a predetermined (nonce) value for data, the Enhanced LCG initializes the LCG state. The final CSRF token or other pseudorandom number is generated by periodically enhancing the LCG state to produce a series of pseudorandom integers that are then translated into ASCII values. In comparison to standard LCGs or independent HMAC_DRBG techniques, the Enhanced LCG has many advantages, including security, effectiveness, and computational efficiency. Though it is not entirely random and has a compromised rate when the attacker discovers the contents of the sensitive state, the Enhanced LCG has some arguments against adopting CTR_DRBG.

The report is structured to comprehensively address the issue of Cross-Site Request Forgery (CSRF) attacks and present a robust solution. The introduction provides an overview of the problem and introduces the proposed solution involving HMAC_DRBG and eLCG. The subsequent section delves into the theoretical foundation of HMAC_DRBG and eLCG, elucidating their roles in generating secure CSRF tokens. Moving forward, the implementation details are discussed, highlighting the utilization of Python and Flask for practical application. The heart of the report lies in the Evaluation section, where an array of statistical tests, including the Chi-squared Test for Uniformity, Auto-Correlation Test, Gap Test, and Serial Overlapping Patterns Test, are conducted to rigorously assess the quality, unpredictability, and independence of the generated CSRF numbers. Scatter plot analysis further reinforces the findings. The Discussion section critically examines the results, highlighting the strengths of the approach while acknowledging its limitations and proposing potential avenues for future enhancements. Finally, the Conclusion summarizes the achievements of the study, emphasizing the robustness of the HMAC_DRBG – eLCG based Anti-CSRF Token method and its contribution to web application security, concluding with an outlook on potential future developments in this domain.

2 LITERATURE REVIEW

2.1 A Review on LCG

In this paper [1], the static password authentication is an efficient and easy technology, however it lacks security owing to frequent password changes. Lamport pioneered one-time password (OTP) authentication using hash algorithms in 1981. OTP is a security approach that can be used only once and has a limited time limit, making it appropriate for web and Android-based systems. OTP schemes are rated based on their ease of use, security, and efficiency. The goal of this research is to create a method for generating OTP that is secure, using the Advanced Encrypted Standard (AES) algorithm for encryption and the Linear Congruential Generator (LCG) for random 6 character OTP values.

This paper [2] discusses about the growing internet privacy concerns, data security in IoT applications which is becoming increasingly problematic. The Pseudorandom bit generator (PRBG) is an important component in managing user privacy in IoT devices. The PRBG is considered random if it passes the fifteen standard statistical tests developed by the National Institute of Standards and Technology (NIST). Pseudorandom bits can be generated using a variety of techniques, including the linear feedback shift register (LFSR), the linear congruential generator (LCG), the Blum blum shub generator (BBS), the Coupled linear congruential generator (CLCG), and the dual-coupled linear congruential generator (DCLCG). The LFSR is a basic approach that uses flip flops and an XOR gate, however due to its linearity structure, it fails randomization tests. The LCG has a smaller size and less hardware complexity, but it is more expensive.

Within this paper [3] the linear congruent method (LCM) is explained to be a well-known pseudo-random number generating algorithm that produces predictable outcomes. Researchers have created versions and hybrids of LCG, such as dual-CLCG, that provide better-randomized outcomes but require a higher amount of processing. These adjustments are popular in cryptography and are safer for randomizing queries. Hybrid LCG is used to encrypt data, randomize exam questions, and generate more random passwords. CLCG and dual CLCG are also used in hybrids with RSA and ECC to increase encryption key complexity. This research suggests a modified LCG with an inverse element, which would randomize outcomes more but would be less complex than CLCG and dual-CLCG calculations.

In this paper [4] identified that LCG has known issues. The security and privacy of data in IoT applications are crucial, as millions of devices generate big data. Common low-complexity PRBGs like LCG and LFSR fail randomness tests and are insecure due to their linearity structure. A new PRBG method, the modified dual-CLCG, uses XOR logic, generates 2n pseudorandom bits, passes all fifteen NIST benchmark tests, and can be implemented using Verilog HDL and prototyped on commercially available FPGA devices.

2.2 A Review on DRBG

In this paper [5] to create keys, nonces, and initialization vectors, cryptographic systems require a considerable amount of randomness. Because many computers lack the high-quality physical randomness required to generate these values, pseudorandom number generators (DRBGs) are used to stretch small bits of actual unpredictability into huge volumes of pseudorandom output. Compromise of a generator can jeopardize the security of nearly any cryptosystem built on it, making DRBGs critical to security. Compromising a generator might be disastrous since an adversary who can foresee future generator outputs may be able to forecast private keys or recover long-term keys used as input to protocol execution. Validation standards now rely mostly on statistical tests and test vectors, neither of which guarantees that the output is pseudo-random. Backdoored PRGs that are undetectable by black-box testing are a possibility. DRBGs have not gotten the attention they deserve, with numerous fatal faults discovered at both the design and implementation levels. Some defects are unintentional, while others are intentional. The NSA's purported backdooring of the Dual EC DRBG standard is a critical flaw in DRBG deployment. Current government guidelines encourage certain designs but lack discipline, with some DRBG designs becoming industry standard. There are no formal security proofs for these algorithms, and the design methods are neither open nor rigorous.

In the paper [6] the Deterministic Random bit generators (DRBGs) are explained to be a crucial cryptographic primitives used in key generation and authentication protocols in encryption systems. A randomness source provides entropy input to DRBGs for the seed, and the security of DRBG is dependent on the secrecy of seeds. PUFs, which are physical fingerprints of Integrated Circuits (ICs), can have unclonable and unpredictable properties, making it difficult for attackers to steal secrets from them. SRAM PUFs, which can be implemented using intrinsic variation, are the most common PUF implementation. Noisy cells are appropriate for DRBG seeds because they are unpredictable, easily incorporated in a cryptographic module boundary, and have sufficient entropy. This enables the use of SRAM PUFs as PRNG seed sources. However, not all SRAM chips are capable. However, not all SRAM chips can be employed as entropy sources in every circumstance. To monitor entropy in SRAM start-up patterns online, a Get Entropy module is proposed. The study develops a streaming random number generator in accordance with NIST SP800-90 requirements, generating pseudorandom numbers with full entropy seeds.

In this work [7] the nondeterministic random bit generators (NRBG) generate random numbers utilizing unexpected physical sources of randomness, whereas deterministic random bit generators (DRBG) generate sequences of numbers using internal state values. NIST has released a number of deterministic random number generators, including the Dual EC DRBG, which has been discovered to be vulnerable. Cryptographic specialists have pointed out problems in DRBG, although grasping these flaws might be difficult for students or professionals who do not have extensive cryptography knowledge. This study presents proof of concept for the vulnerability and suggests mitigation strategies. Because NIST cryptographic standards, including DRBGs, are widely used and implemented in cryptographic solutions, any criticism or reported weaknesses must be thoroughly investigated before implementation.

2.3 A Review on HMAC

This paper [8] illustrates, Message authentication codes (MAC) are increasingly being used in digital communication systems to assure data integrity. HMAC is a popular cryptographic hash function that executes faster than block-cipher techniques. The National Institute of Standards and Technology (NIST) established the most extensively used SHA family, which has been improved and refined throughout time. Because traditional cryptanalysis techniques are difficult to break, cryptographic hardware is vulnerable to circuit-level attacks. A typical method, side channel analysis, can recover the key by collecting and analyzing power, electromagnetic, and temporal data. To protect cryptographic hardware, common countermeasures against side channel analysis must be considered during hardware design.

This work [9] describes a scan-based attack against HMAC-SHA-256, a popular message authentication mechanism. The method examines scan data from a scan chain and uses it to recover secret information. It isolates 64 bit-transition groups, groups them into 32 pairs, and decides whether each pair corresponds to internal register an or e. The method successfully recovers two secret keys from scan data, even when the scan chain contains registers different than those of the HMAC-SHA-256 circuit.

In this paper [10] the Cloud computing is described to a cutting-edge technology that offers continuous service with minimum complexity, making it appropriate for a wide range of applications like social networking, financial solutions, and ecommerce. However, some solutions may necessitate the revelation of secret and private information, thereby posing privacy concerns for both organizations and end users. To address these concerns, academics are investigating cloud security and privacy issues during data exchange. Third Party Auditor (TPA) is an examination process used to guarantee customer data confidentiality and privacy. DNA cryptography is a frequently used encryption method. TPA generates a Hash-based MAC (Message Authentication Code) of ciphered data stored in the cloud server, and users send HMAC values to TPA using the same secret key for data integrity verification.

This paper [11] addresses standard pseudo-random (PRNG) methods such as BBS, LCG, and Dual EC DRBG. Though BBS PRNG is appropriate for cryptography applications, it is slow pace & computationally expensive. LCG is an algorithm which employs a discontinuous piecewise linear equation to generate a sequence of pseudo-random numbers. Dual EC DRBG is an elliptic curve cryptography technique that uses an CSPRNG deployment. This is, however, inefficient and is frequently criticized for shortcomings in security. The study presents a modification to Dual EC DRBG by integrating BBS, LCG, and modified EC DRBG to form the BEL CSPRNG combined the strategy. The strategy has been verified and contrasted against standard PRNGs involving LCG, BBS, & modified EC DRBG employing the NIST test suite. The paper closes by comparing the output with traditional PRNGs & and the modified EC DRBG.

This study of the literature looks at methods and strategies for pseudorandom number generation, authentication, internet privacy problems, and cryptographic security. The Enhanced LCG methodology is introduced, a revolutionary way combining HMAC_DRBG and LCG that demonstrates its security and efficiency benefits. The implementation section addresses the Enhanced LCG's practical application for creating Cross-Site Request Forgery (CSRF) tokens in web applications, emphasizing its contribution to upgrading CSRF prevention methods.

Statistical tests show that generated CSRF numbers utilizing HMAC_DRBG-seeded eLCG are of high quality, uniform, and independent. Several statistical tests, including chi-squared, auto-correlation, gap, and serial overlapping patterns tests, validate the approach's randomness and effectiveness.

Finally, the works under consideration contribute to the advancement of pseudorandom number generation, authentication methods, and cryptographic security. The Enhanced LCG methodology looks promise for improving CSRF token generation and web application security. More research and development are required to solve new difficulties and maintain the security of modern digital settings.

Upon reviewing all these papers, the LCG has major issue in generating random numbers, however when modified the way LCG and enhancing the LCG by making it unpredictable for any attackers to avoid brute force.

3 RESEARCH METHODOLOGY

3.1 JUSTIFICATION

Three different forms of PRNG techniques were suggested by the NIST SP 800-90A recommendation: Hash_DRBG, HMAC_DRBG, and CTR_DRBG. The PRNG technique used in this study is HMAC_DRBG, and the method was elected considering a couple of justifications.

3.1.1 Logic to avoid adopting CTR_DRBG

Encryption blocks serves to be the building blocks in CTR_DRBG. Random permutation constitutes the primary element in an encryption block.
Function f: $\{0,1\}^p$ $\{0,1\}^p$ is seen called a random function if:
1) Every the range and domain values being alike;
2) The domain to range mapping action is bijective; and
3) Each domain value maintains the same probability of having been picked by every domain value.

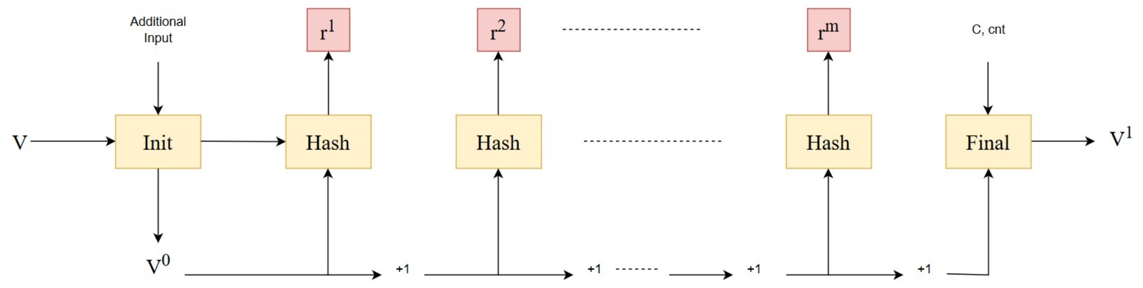
The encryption block is inappropriate for producing the random number due to the random permutation attribute. Let's say that a PRNG based on an encryption block is employed for creating a series of four-digit random integers that vary between 0 to 9, where:

• The encryption blocks employed in this case is the random permutation function
E(k,): $\{0,1\}^n$ x $\{0,9\} \cap$ N $\{0,9\} \cap$ N with k ϵ $\{0,1\}^n$
• The PRNG uses the following algorithm to produce four random numbers:
E(k, r + 1)||E(k, r + 2)||E(k, r + 3)||E(k, r + 4)
with r ϵ $\{0,9\} \cap$ N

A sequence produced by the PRNG will undoubtedly contain unique numbers for each digit. Therefore, from a total of 10,000 sequences generated by 10 x 10 x 10 x 10, there are a total of 5,040 combinations of four-digit sequences that can be formed. This massive reduction in sample space size due to block encryption is by not entirely random.

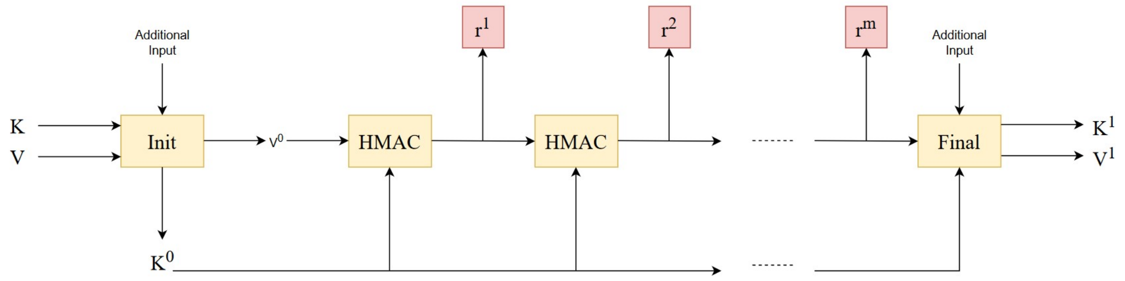
3.1.2 Reason not to select Hash_DRBG

When compared to Hash_DRBG, HMAC_DRBG has two key advantages:
• the compromised rate when the attacker learns the contents of the sensitive state; and
• the originality of the data delivered by the HMAC algorithm compared to the value that is generated by the hash function's output.



The method of generating function for Hash_DRBG is shown in Figure 2. If the attacker has the counter value of the kth transition state, or $V^k = V^n + k$ for $k \epsilon$ [1,m], the attacker is able to recover the initial counter value V^0 and utilize it to steal all of the hash block outputs within the same generate function call, before as well as after the k^{th} transition.

The generating function technique for the HMAC_DRBG is shown in Figure 3. When the attacker obtains the starting key value K^0 as well as the result value of the k^{th} transition state, or r^k, the attacker is able to calculate the next block results utilizing the same generating function call, particularly r^j = HMAC(K^0, r^{j-1}) where j ϵ [k+1,m]. Even though the attacker already has the K0 value, restoring the results of the blocks produced prior to the k^{th} transition state is challenging because the attacker must carry out a preimage attack using the r^j value, where r^j = HMAC(K^0, r^{j-1}) for j ϵ [1,k], in order to retrieve the r^{j-1} output block.



It is commonly understood that the hash function is a deterministic function, meaning that for a given input value, the output value will always be the same. The HMAC function is also deterministic because it utilizes the hash function as one of its building blocks. The secret key utilized by the HMAC function is one way that it differs from the hash function. If the secret key supplied on both of the inputs is unique, two identical data inputs will result in two independent data outcomes. Figure 3 demonstrates that the key K^0 is always updated at the conclusion of the HMAC_DRBG function call. As a result, every call to the create function with a counter value V will end up resulting in a different random number. [12]

The Reason to make LCG into enhanced LCG is because upon revieing all papers above we can conclude that seed value of the LCG makes it so predictable. Hence the in traditional LCG the output is feed back as the seed value of the next iteration. Hence to resolve this in my approach the Enhanced LCG will not use the previous output however the seed value would be given from HMAC_DRBG.

3.1.3 Justification Based on Objectives Chosen

Objective 1: Using Python, for Implementing the HMAC-DRBG - eLCG-Based algorithm, modified the traditional LCG and combined it with HMAC_DRBG to produce random numbers and random numbers are then converted to ASCII to generate random CSRF Tokens.
Objective 2: Using the chi-squared test, determine whether the distribution of created CSRF tokens is constant across a certain range.
Objective 3: For an evaluation of the independence of the CSRF token pattern, computing auto-correlation coefficients for multiple delays.

Objective 4: To estimate the average gap between consecutive CSRF tokens, use the gap test and compare it to the expected gap based on a uniform distribution. To find and evaluate instances of specified patterns within the CSRF token sequence, use the serial overlapping patterns test.

This project has used Python for implementing all these above objectives.

4 DESIGN

4.1 TRADITIONAL LCG & ENHANCED LCG

The LCG approach establishes a recurrence relation wherein a remainder division strategy is utilized to calculate the subsequent random number depending on the last one that was generated. It is a basic pseudo-random number generator. If the values of the parameters selected are not adequate ones, this technique could end up in a repeated pattern of the numbers created. [13]

The First Number of LCG would be generated using below,

X_1 = $(aX_0 + c) \bmod m$, where initial X_0 is the seed value. c, a & m is selected and assumed with some values & they are called LCG parameters.

The second number of the series would generate using

$X_2 = (aX_1 + c) \bmod m$

.

.

And Finally the last number of the series would be

$X_n = (aX_{n-1} + c) \bmod m$

Like we discussed earlier patterns of the number generated in LCG would repeat, after some iteration.

The Enhanced LCG (eLCG) is a Combination of HMAC_DRBG and LCG & its an novel technique for generating CSRF Token & its diagram is shown in the figure 4. When compared to traditional LCG here the enhanced LCG will not be using its previous value X_{n-1} to generate its next number X_n however it will be using HMAC_DRBG's V^1 output would be feuded as the seed value for the each iteration in the enhanced LCG. The Output of Enhanced LCG would be saved as X= $\{X_1, X_2, X_3 \dots \dots \dots X_n\}$ where X_n would be a 10 or any digit number as per the application requirement. This 16 digit number would then be converted to ASCII to form as an CSRF TOKEN.

The First number of the eLCG would be generated using

$X_1 = (aV^1 + c) \bmod m$, Where V^1 latest output of the HMAC_DRBG

Since as per [14] would be using a = 1103515245, c = 12345 & m = 2**32 so that we could get big number as output for an anti-CSRF Token.

The Second Number of eLCG would be generated using,

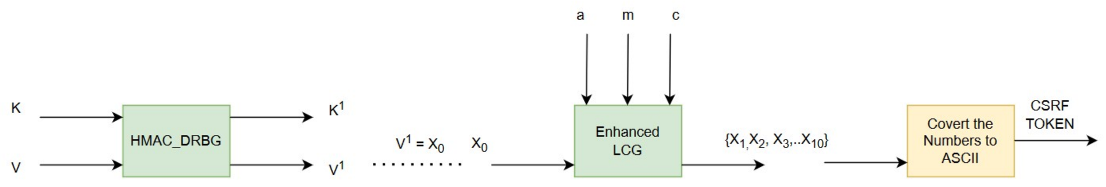
$X_2 = (aV^2 + c) \bmod m$, where V^2 latest output of the HMAC_DRBG,

.

.

$X_n = (aV^n + c) \bmod m$, where again V^n is latest output of HMAC_DRBG

Hence HMAC_DRBG will loop the iteration for the traditional LCG's formula, where each time the seed value X_{n-1} is updated by the HMAC_DRBG's output V^n. So in eLCG the V^n is not the previous eLCG output but HMAC_DRBG's new output value.
`



4.2 HMAC_DRBG

4.2.1 Overview

HMAC_DRBG (Hash-based Message Authentication Code Deterministic Random Bit Generator) is a cryptographic pseudorandom number generator. It uses the HMAC (Hash-based Message Authentication Code) design to generate random output by combining a cryptographic hash function and a secret key.

A seed, a nonce, and a secret key are used to initialize the HMAC_DRBG algorithm. The seed and nonce are used to determine the generator's initial state. The generator then generates random bits by performing the HMAC operation to its internal state periodically. It permits the injection of supplementary entropy, known as the entropy input, during the reseeding procedure to increase the security of HMAC_DRBG. To keep the random output unpredictable while updating the internal state, periodic reseeding is necessary. The security of HMAC_DRBG is based on the robustness of the underlying cryptographic hash function & the confidentiality of the secret key. As long as these components are secure, HMAC_DRBG is tolerant to prediction attacks, guaranteeing the randomness of the generated bits. Applications & cryptographic protocols requiring high-quality random numbers can use HMAC_DRBG. It provides a trustworthy supply of random bits, which is essential for maintaining the privacy and reliability of cryptographic operations as well as protection of private data in modern safety systems. [15]

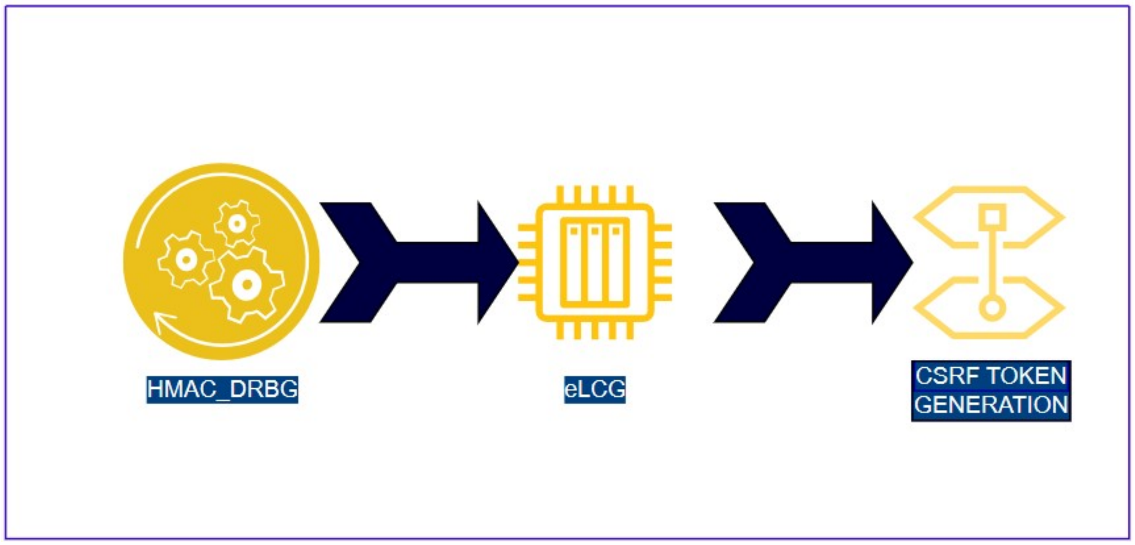
4.2.2 HMAC_DRBG Algorithm

HMAC_DRBG (Hash-based Message Authentication Code Deterministic Random Bit Generator) is a pseudorandom number generator which employs a cryptographic hash algorithm & a secret key to generate random numbers. Various steps are involved in the generation process to assure reliability & randomness.

1. **Initialization:** HMAC_DRBG is set up by supplying it a seed, a nonce, & the secret key. The seed & nonce will be utilized to establish the generator's initial state.
2. **Entropy Input:** To improve the generator's security, supplemental entropy can be added while in the reseeding phase. Such supplementary feed is commonly referred to as the entropy input.
3. **Reseeding:** It is critical for preserving the security of HMAC_DRBG on a regular basis. The generator is reseeded with new entropy input & a new nonce after generating a specific number of random bits or determining an imminent security issue.
4. **Pseudorandom Bit Generation:** Once initialized, HMAC_DRBG generates pseudorandom bits via constantly conducting the HMAC operation to the internal state. The produced bits are retrieved using the HMAC function output.
5. **Prediction Resistance**: HMAC_DRBG has prediction barriers, which indicates that regardless of whether an attacker monitors certain preceding output, he can't predict what is to come without understanding the internal state & secret key. [15]

4.3 HMAC_DRBG and ENHANCED LCG

The HMAC_DRBG and Enhanced LCG is an innovative method that combines the best features of both HMAC_DRBG & LCG to generate a safe and effective pseudorandom number generation procedure as shown in the figure 1. The HMAC_DRBG & Enhanced LCG strikes an acceptable solution between cryptographic safety and efficiency by combining these two approaches, resulting in it being ideal for services needing robust pseudorandom number generation, including the compilation of anti-CSRF tokens in web applications.



HMAC_DRBG Seed Generation: The procedure begins with the generation of a cryptographically safe random seed utilizing HMAC_DRBG. This starting point provides unpredictability & serves as the base for the two distinct HMAC_DRBG and LCG phases of this approach.

HMAC_DRBG Initialization: The HMAC_DRBG algorithm uses the produced seed & a predefined (nonce) value for the data (typically zero) as inputs. It generates a pseudorandom number through the combination of HMAC and a secure hash function, for instance SHA-256. This pseudorandom number operates as the LCG's initial state.

eLCG State Initialization: The pseudorandom result generated by HMAC_DRBG acts as the eLCG's initial state. The eLCG is a straightforward technique that creates a series of pseudorandom integers utilizing a linear recurrence relation. The eLCG parameters, which include a multiplier and an increment, govern how the state changes to generate successive pseudorandom numbers.

Token Generation Using eLCG: The eLCG state is regularly Enhanced in order to produce a series of pseudorandom numbers. Each number is then converted into an ASCII value that corresponds to that number. These characters are then concatenated to generate the final CSRF token or any other pseudorandom number that the application requires. [12]

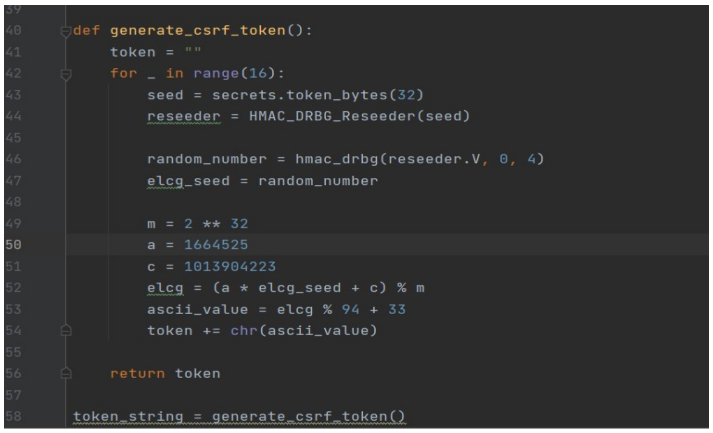
5 IMPLEMENTATION

In this study, we demonstrate how to generate and validate Cross-Site Request Forgery (CSRF) tokens using the Flask web application. Web security flaws known as CSRF attacks allow an attacker to force the browser of a user to execute improper activities at a reliable website. We have created a strong CSRF prevention mechanism utilizing encrypted token generation and validation mechanisms to protect against such attacks.

5.1 TOOLS USED

Python: Python is a high-level programming language with dynamic semantics, making it suitable for RAPID and scripting. Its straightforward syntax prioritizes readability while also lowering maintenance costs and supporting modules and packages. Python's interpreter and standard library are available in both source and binary versions, and it is frequently used to boost productivity. The cycle of edit-test-debug is quick, and debugging is simple using a source-level debugger. The fast edit-test-debug cycle, on the other hand, can be a speedy way to debug a program, making it an effective strategy. [16]

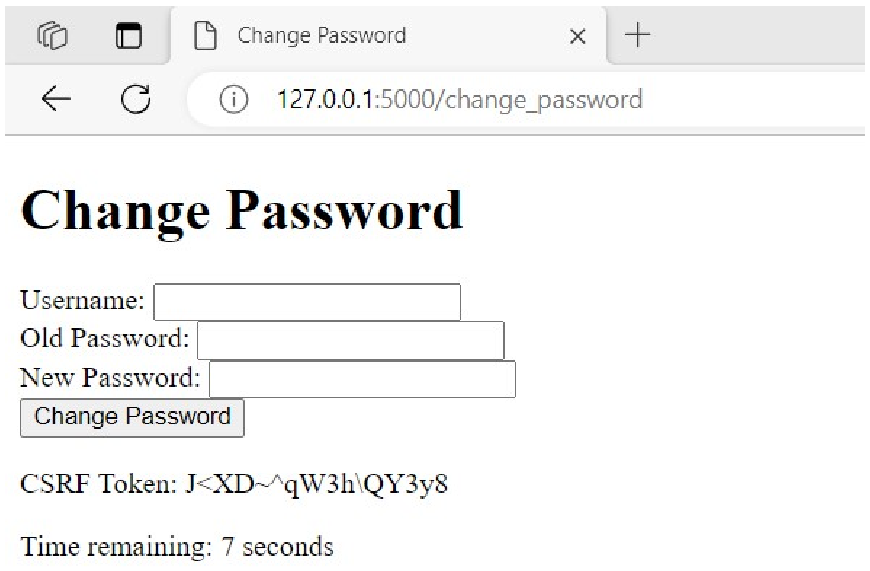
The Implementation of this revolutionary idea has been completed with aid of Python. Below is the snap shot of the small python code of the logic behind the CSRF Token generation using HMAC_DRBG and eLCG.



```python
def generate_csrf_token():
    token = ""
    for _ in range(16):
        seed = secrets.token_bytes(32)
        reseeder = HMAC_DRBG_Reseeder(seed)

        random_number = hmac_drbg(reseeder.V, 0, 4)
        elcg_seed = random_number

        m = 2 ** 32
        a = 1664525
        c = 1013904223
        elcg = (a * elcg_seed + c) % m
        ascii_value = elcg % 94 + 33
        token += chr(ascii_value)

    return token

token_string = generate_csrf_token()
```

Flask: Flask is a Python web framework that allows for simple application development. It is a microframework that lacks an ORM but has capabilities such as URL routing and a template engine. Flask, created by Armin Ronacher, is built on the Werkzeg WSGI toolkit and the Jinja2 template engine, both of which are part of the Pocco project. [17]

The Flask has been used here in this project to host an small web application & to show the CSRF Token generation on the hosted web app. Below is the snapshot of the website running in python flask.



Change Password

Username:
Old Password:
New Password:
Change Password

CSRF Token: J<XD~^qW3h\QY3y8

Time remaining: 7 seconds

MySQL Work Bench: MySQL Workbench makes database design and maintenance easier, automates chores, and enhances communication between DBA and developer teams. It allows data architects to visualize requirements, communicate with stakeholders, and address design difficulties prior to making large investments. It offers model-driven database architecture, allowing for adaptability to changing business requirements. Model and Schema Validation utilities enforce recommended practices for data modeling as well as MySQL-specific physical design standards, guaranteeing that no errors are committed when creating new ER diagrams or physical databases**.** [18]

The MySql Database has been used here to store the username & password of the application.

5.2 Overview of CSRF Token Generation and Validation Implementation

Flask is a web framework designed with Python for our implementation. The steps used in this implementation are as follows:

1. **HMAC_DRBG-based Token Generation**: HMAC_DRBG (Hash-based Message Authentication Code Deterministic Random Bit Generator) serves to generate a secure and unpredictable seed value for eLCG. This process assures that each produced number is random, thus rendering it impossible for attackers to foresee or alter.
2. **Enhanced Linear Congruential Generator (eLCG)**: Based on the HMAC_DRBG seed, we apply eLCG, a pseudo-random number generator, to produce a sequence of

numbers. This sequence serves as the foundation for our CSRF token, which is subsequently processed to yield a 16-character alphanumeric token.

3. **Database Integration**: User authentication and password management are included in the implementation. We connect to a MySQL database to check user credentials and securely update passwords.
4. **Session Management**: We use Flask's session functionality in order to preserve the CSRF token validity. The CSRF token and its expiration are saved in the session, enabling designers to keep record of the document's generation and date of expiration.

5. **API Endpoint Security:** We set up an additional inspector called **validate_api_csrf_token** for API routes which need CSRF protection. This inspector validates API requests for the presence & validity of the X-CSRF-Token header.

This implementation's primary outcome is a strong CSRF token which defends against CSRF attacks. Whenever clients access the program, they are provided with a session-specific token. The corresponding token is used to authenticate further input from forms & API queries, to guarantee only those with permission can conduct actions on the website.

The CSRF token synthesis & authentication methods are rigorously tested to assure their effectiveness. We use several test scenarios to analyze the following aspects:
1. Token Unpredictability: We ensure that the newly created CSRF tokens have an elevated degree of unpredictability, which renders them resistant to guessing or brute-force attacks.
2. Token Expiration: The session-based CSRF tokens are examined for security reasons they exhaust following a set time (e.g., 30 seconds). This eliminates the repetitious use of outdated tokens therefore increases protection.
3. API Endpoint Protection: API endpoints controlled by the CSRF token validation decorator are assessed to ensure that illegitimate attempts are appropriately refused.

6 EVALUATION

This evaluation aims to generate a sequence of Cross-Site Request Forgery (CSRF) numbers using HMAC_DRBG (HMAC-based Deterministic Random Bit Generator) with the results to answer the research question. The CSRF numbers are generated using HMAC_DRBG-seeded eLCG. So basically once eLCG give an output it will be in the form number and these numbers are termed as **CSRF Numbers** (CSRF) & then these numbers are converted to an alpha numeric with some special charter to a CSRF TOKEN. Which includes statistical tests to evaluate the quality and randomness of the generated CSRF numbers.

CSRF Number Generation with HMAC_DRBG

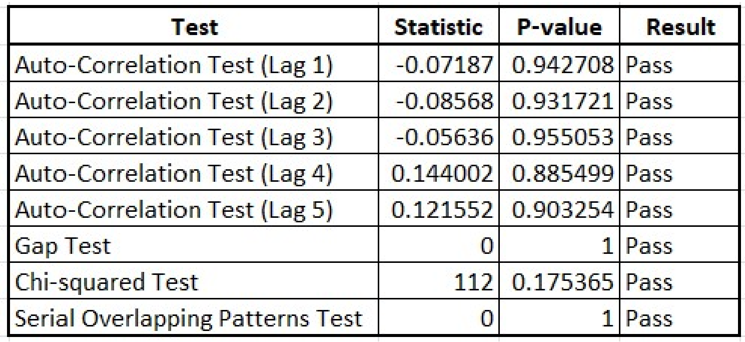
The procedure of generating a CSRF number kicks off by accumulating a 32-bit seed from a secure random source utilizing the os.urandom function in python. HMAC_DRBG is subsequently employed in order to produce pseudo-random numbers based on the seed. HMAC_DRBG uses HMAC as the hash function with SHA-256 to ensure that all generated numbers remain private and unique. The HMAC_DRBG results are utilized as the seed value for the eLCG.

eLCG with HMAC_DRBG Seed

The Modulus m = 2**32, multiplier a = 1103515245, and increment c = 12345 are the eLCG parameters & these parameters are choses using [14]. The eLCG generates an array of series depending on the HMAC_DRBG seed. The eLCG formula would be the same as the LCG formula, but it would not use its own seed value, as mentioned in the design section earlier. $X_n = (a\mathrm{V}^{n-1} + c) \bmod m$, where V^n is the last number produced via HMAC_DRBG.

6.1 RESULTS AND ANALYSIS

Upon producing the CSRF numbers to perform the statistical tests, the results are examined to determine the sequence's quality. The statistical results of the generated CSRF Number sequences are displayed in the table below.



Test	Statistic	P-value	Result
Auto-Correlation Test (Lag 1)	-0.07187	0.942708	Pass
Auto-Correlation Test (Lag 2)	-0.08568	0.931721	Pass
Auto-Correlation Test (Lag 3)	-0.05636	0.955053	Pass
Auto-Correlation Test (Lag 4)	0.144002	0.885499	Pass
Auto-Correlation Test (Lag 5)	0.121552	0.903254	Pass
Gap Test	0	1	Pass
Chi-squared Test	112	0.175365	Pass
Serial Overlapping Patterns Test	0	1	Pass

6.1.1 STATISTICAL TESTS

As shown in the above table several statistical tests are carried out on the sequence to assess the overall quality and randomness of the obtained CSRF numbers. Following are the detailed test analysis. For all the tests the observed value will be numbers generated first in sequence which would be saved in the excel sheet after running the program. So initially when the statistical test program is executed for the first the program would compute the statical tests however it just stores the Output in the Excel file & this would be utilized as observed value, by all the tests when required.

1. **Chi-squared Test for Uniformity**

The chi-squared test evaluates if the CSRF numbers are split equally throughout the range. The ordered list is divided across a certain amount of bins, and the observed frequencies in each bin examined to the expected frequencies to obtain a uniform distribution. A significant p-value & a relatively small chi-squared value imply that the sequence is uniformly distributed. [19]

The Chi-Squared test is calculated using the below formula.

$$X^2 = \sum\left(\frac{(Observed - Expected)^2}{Expected}\right)$$

The Formula its self explanatory where the X^2 represents the Chi-Square. In this research each bin's Expected frequency is computed as follows:

Expected = (Total number of data points) / (Number of bins)

The chi-squared value in the presented instance is 112.0 as per the Table 1, whereas the corresponding p-value is 0.17536488297985384. The p-value estimates the likelihood of achieving such a high chi-squared value if the numbers were uniformly distributed (random). We fail to reject the null hypothesis because the p-value is bigger than the standard statistical threshold of 0.05, indicating that the pattern is not considerably distinct compared to a uniform distribution. By this instance, the numbers seems to be produced completely random through the chi-squared test.

2. Auto-Correlation Test

Autocorrelation is an essential idea in data analysis that analyzes the correlation between a time series & its lags. This aids individuals in comprehending the connection among the numbers of a variable at different junctures in time. The mathematical equation for autocorrelation resembles to the equation for correlation, with the exception it incorporates lagged forms of the time series. The auto-correlation function (ACF) of the time series "X" containing observations "x_1, x_2..., x_n" at lag "k" is described in the following manner: [20]

$$\text{ACF}(k) = \frac{\sum_{t=k+1}^{n} (x_t - \bar{x})(x_{t-k} - \bar{x})}{\sum_{t=k+1}^{n} (x_t - \bar{x})^2}$$

The auto-correlation test examines the relationship between CSRF numbers at various lags. The auto-correlation estimates are computed for a predetermined number of lags. Low auto-correlation readings recommend that numbers at various points in the pattern are autonomous, signifying the overall sequence lacks expected trends. The correlation between a series to itself over various time latencies can be determined via auto-correlation. This serves to determine if a given sequence features any patterns or dependencies. The auto-correlation findings illustrate the correlation values across multiple lags (1–5). Each one of the autocorrelation coefficients extremely near to zero, while their p-values are large (higher than 0.05). Which implies that the numbers generated contains no solid serial correlation, thereby being another proof of randomness.

3. Gap Test

The gap test is a statistical test that looks at the average number of gaps between consecutive numbers within a sequence to determine their randomness. This is concerned in acknowledging trends or categorizing of values in a certain period or group. If α and β represent two real numbers with $0 \leq \alpha < \beta \leq 1$, we wish to look at the lengths of successive subsequent sequences U_{j+1},..., U_{j+r} where U_{j+r} exists within α and β while the remaining U's do not.

(This r + 1 integer subsequence symbolizes an r-length gap.)

so $g_i = | U_{i+1} - U_i |$, Where g_i represents the gap value between U_{i+1} & U_i

Obtaining the Standardized Gap (s-Gap): The standardized gap (s-gap) is determined by contrasting the mean of the observed gaps to the overall mean of all potential gaps. The ss-gap is calculated using the following formula: [21]

$$\text{s-Gap} = \frac{|\mathit{Mean\ of\ Gaps} - \mathit{Mean\ of\ Observed\ Gaps}|}{\mathit{Standard\ Deviation\ of\ Observed\ Gaps}}$$

The gap test scans for gaps in a series of sequential CSRF numbers. For a random sequence, it measures the observed average gap to the expected average gap. A low normalized gap value signifies the pattern of sequences may have regular trends. The S-Gap is the actual gap within the series, while the predicted gap is a theoretical number calculated using a uniform distribution. The result of the gap analysis returns a p-value of 1.0, demonstrating that that there's no statistically significant distinction among the expected & observed gaps. Which allows credence to the assertion because the sequence is uniformly distributed & random.

4. Serial Overlapping Patterns Test

This is a nothing but the Overlapping test taken from NIST [22]. The serial overlapping patterns test scans the CSRF number series for occurrences of a stipulated pattern. The number of overlapping instances of the pattern is determined & contrasted to the critical value. A significantly elevated chi-squared value & a low p-value imply that the series contains non-random trends.

This test is used to check for the presence of specific patterns within the sequence. The pattern "101" is tested, and it is found that there are no overlapping occurrences of this pattern in the sequence. The critical value for the test is 122.10773460981942, and since the observed value (0) is less than the critical value, it suggests that the sequence does not contain any significant occurrences of the given pattern.

5. Uniformity and Independence

For cryptographic applications, independence and uniformity are crucial characteristics. Statistical tests are provided in NIST's 800-22 Revision 1a to assess the reliability and applicability of RNGs and PRNGs. [22]

The chi-squared and auto-correlation analyses convey details regarding the uniformity & independence of the CSRF numbers. A strong chi-squared test p-value along with low auto-correlation results hint that the sequence exhibits excellent uniformity & independence, demonstrating how the HMAC_DRBG-seeded eLCG creates a random sequence with significant efficiency.

6. Gap Test and Serial Overlapping Patterns Test

Statistical tests for RNGs and PRNGs that assess their uniformity and independence include the Gap Test and Overlapping Patterns Test. These tests are successful if the generator generates sequences that are less predictable or biased, making them appropriate for security-sensitive situations. [22]. The gap test and serial overlapping patterns test reveal any trends or consistency in the sequence. In the serial overlapping patterns test, a low averaged gap value along with a large chi-squared value and a low p-value signal an existence of non-random patterns. In order to produce useful findings, the analysis must take the significance threshold & sample size into account.

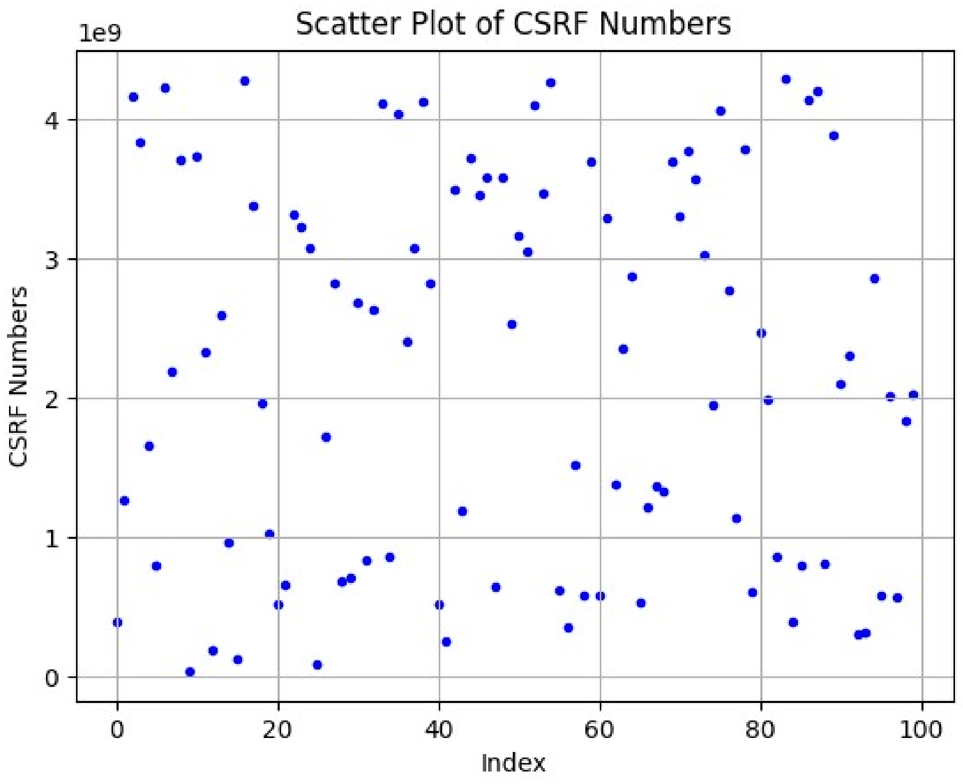
The use of HMAC_DRBG and eLCG for CSRF number production presents a reliable and safe method of developing random CSRF numbers. The statistical tests support the sequence's quality inherent randomness. The usage of HMAC_DRBG permits the creation of an attack-resistant pattern ideal to be utilized as CSRF tokens in web applications.

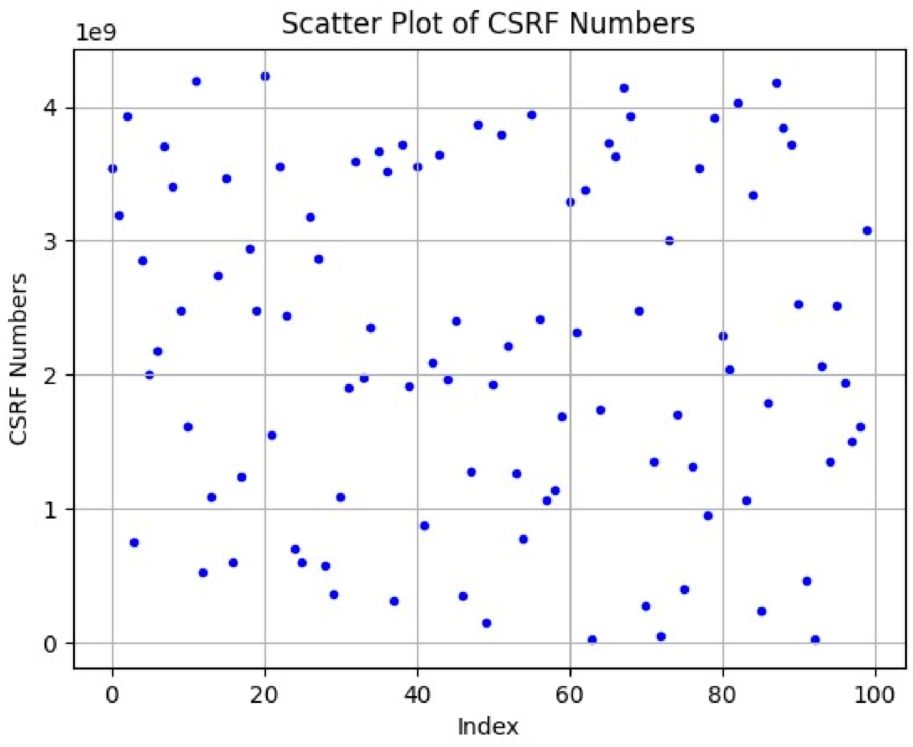
The generated CSRF numbers and the statistical test results are saved to an Excel file for further analysis and record-keeping. By using HMAC_DRBG-seeded eLCG, this implementation enhances the security of CSRF tokens and contributes to the overall security of web applications.

6.1.2 SCATTER PLOT ANALYSIS

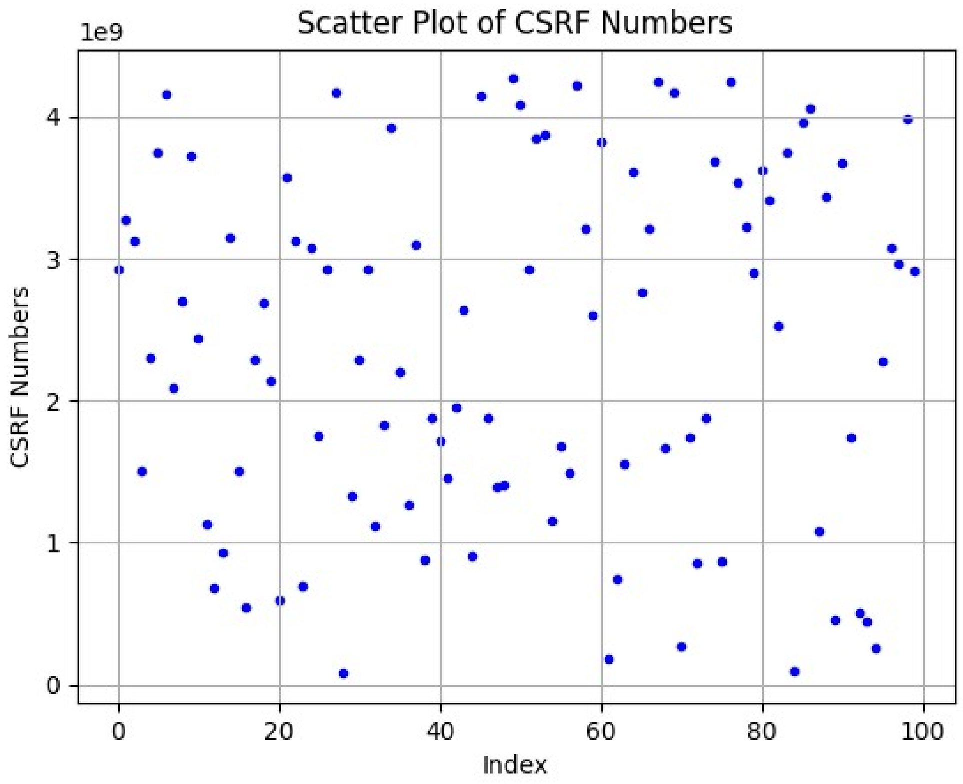
I have generated three sets of CSRF numbers using the revolutionary algorithm and carefully evaluated their distributions using scatter plots. Each scatter plot showcases the generated numbers, with the x-axis representing the index of the number in the sequence, and the y-axis representing the actual value of the number.

Scatter Plot 1




Scatter Plot of CSRF Numbers
1e9
CSRF Numbers
Index
0
1
2
3
4
0
20
40
60
80
100

Plot 3:


Scatter Plot of CSRF Numbers
1e9
CSRF Numbers
Index
0
1
2
3
4
0
20
40
60
80
100

6.1.3 Key Observations

Upon a detailed evaluation of the above scatter plots, the following observations are made:

Distinct Points: In all the three scatter plots, each dot represents a unique CSRF number. Remarkably, there are no dots overlap between any of the dots within the single scatter plot. This absence of overlapping points indicates that the generated numbers exhibits a high level of uniqueness and variability.

Uniform Distribution: Across all the above scatter plots, the dots are uniformly spread out without any discernible clusters, trends, or patterns. This uniform distribution implies that our revolutionary algorithm generation process is effectively producing CSRF numbers & which in turn producing a effective CSRF Token for the websites designed.

Lack of Correlation: Consecutive CSRF numbers show no evident correlation, as indicated by the lack of discernible trends or directional patterns in the scatter plots. This independence between consecutive numbers further reinforces the randomness and quality of our generation method.

6.2 Implications and Significance

The absence of overlapping dots in our scatter plots would be a strong indicator of the robustness & randomness of our CSRF number generation process. These observations would have several important implications as mentioned below:

Security: The uniqueness & lack of overlap of the dots in the plots suggest a low likelihood of predictability, enhancing the security of our CSRF Token against unauthorized access & CSRF attacks.

Integrity: The uniform distribution & lack of correlation enables to realize that generated CSRF numbers can be effectively distributed & utilized across various other contexts without introducing vulnerabilities or biases, for example where ever this algorithm could be implemented.

Reliability: The thorough analysis of the scatter plots provides additional confidence in the quality of our CSRF Token generation, contributing to the reliability and stability of our web applications.

7 DISCUSSION

This in-depth discussion delves into the results of the experimentation and analysis of HMAC_DRBG – eLCG based Anti-CSRF Token method, with the goal of critically assessing its efficacy in website security and CSRF avoidance. The experiment produced good results, highlighting the advantages of the eLCG-based approach. Statistical tests were performed on the generated CSRF numbers to determine their quality, unpredictability, homogeneity, and independence.

The eLCG-based Anti-CSRF Token technique creates safe and unpredictable CSRF tokens that have strong independence, uniform distribution, and high unpredictability. By strengthening defenses against CSRF attacks, this technique improves the security of online applications. To confirm the randomness and irrationality of the generated numbers, additional experiments and research are required.

However, the paper notes shortcomings, such as the mechanism's vulnerability to prospective attacks. To react to evolving threats, as with any security solution, continual research and proactive actions are essential. Other complementary studies could provide a more comprehensive view of the mechanism's strengths and shortcomings, as the study concentrated on a specific set of tests and analyses.

Finally, the eLCG-based Anti-CSRF Token technique has demonstrated potential in producing secure and unpredictable CSRF tokens. Statistical analysis and scatter plot observations show that the combination of HMAC_DRBG and eLCG produced a sequence of integers with uniformity, unpredictability, and independence. By offering a strong protection against CSRF attacks, this technique has the potential to significantly contribute to online application security. However more tests could probably conducted to show more evidences that the numbers generated are unpredictable and random in nature. If there was more time given possibly could have tried implementing more statistical tests.

8 CONCLUSION AND FUTURE WORK

This research as per the statistical results one could easily say that this method is more efficient in generating CSRF Tokens which basically answers the :

Research Question: How effectively are anti-CSRF tokens created with a linear congruential generator to prevent CSRF in websites?

The purpose of this research was to solve the important concern of Cross-Site Request Forgery (CSRF) assaults by combining the Enhanced Linear Congruential Generator (eLCG) with HMAC_DRBG. The study proved that using HMAC_DRBG-seeded eLCG to generate CSRF tokens is an efficient and dependable way. Statistical testing proved the generated CSRF numbers' excellent quality, homogeneity, and independence, while scatter plot analysis showed the tokens' uniqueness and unpredictability.

The Anti-CSRF Token system, which is based on eLCG, adds a strong layer of defense against CSRF assaults in web applications. The use of HMAC_DRBG increases unpredictability, making it extremely difficult for attackers to exploit weaknesses. This method not only protects user data but also improves the overall security posture of online apps.

The research does have limitations, however, because its efficacy is strongly dependent on the proper implementation and management of the underlying cryptographic components. Furthermore, for full validation, the evaluation may need to be broadened to accept larger and more diverse datasets.

Future studies will provide promising options for improving and expanding on the current findings. This mechanism's use can be expanded to popular cloud platforms such as Google Cloud Platform (GCP) and Microsoft Azure, reinforcing authentication processes and providing a more robust defense against cyber threats. A subsequent research project might investigate adaptive ways to token production based on dynamic threat assessments, which would involve constant monitoring of application activity and user interactions in order to alter the token generation process in real-time.

Furthermore, investigating the incorporation of hardware-based random number generators for seed creation could add an extra layer of protection.

In conclusion, this research has given a fresh and practical answer to the ongoing problem of CSRF attacks. The next step is to extend the mechanism's reach to cloud platforms and investigate adaptive token generation approaches. The cybersecurity landscape may be reinforced against emerging threats and vulnerabilities by constantly refining and expanding on this novel strategy.

9 BIBLIOGRAPHY

[1] Imamah, “One Time Password (OTP) Based on Advanced Encrypted Standard (AES) and Linear Congruential Generator(LCG),” Batu, Indonesia, 2018.

[2] A. N., C. U. Kumari, K. Swathi, T. Padma and P. Kora, “Implementation of Modified Dual-Coupled Linear,” Hyderabad, India, 2021.

[3] S. M. S. Irawadi and A. P. Baiq Desy, “Modify Linear Congruent Generator Algorithms Using Inverse Elements of Modulo Multiplication for Randomizing Exams,” Pangkalpinang, Indonesia, 2022.

[4] K. Panda Amit and K. Chandra Ray, “Modified Dual-CLCG Method and its VLSI Architecture for Pseudorandom Bit Generation,” in *IEEE*, 2019.

[5] K. Q. Ye, M. Green, N. Sanguansin, L. Beringer, A. Petcher and A. W. Appel, “Verified Correctness and Security of mbedTLS HMAC-DRBG,” Newark, United States, 2017.

[6] S. L. B. Chen, “A Dynamic Reseeding DRBG Based on SRAM PUFs,” in *IEEE*, 2016.

[7] M. Amin and M. Afzal, “On the vulnerability of EC DRBG,” in *IEEE*, 2015.

[8] Z. He, L. Wu and X. Zhang, “High-speed Pipeline Design for HMAC of SHA-256 with Masking Scheme,” in *IEEE*, 2018.

[9] D. Oku, M. Yanagisawa and N. Togawa, “A robust scan-based side-channel attack method against HMAC-SHA-256 circuits,” in *IEEE*, Berlin, 2017.

[10] A. Kumar, “Framework for Data Security Using DNA Cryptography and HMAC Technique in Cloud Computing,” in *IEEE*, 2021.

[11] K. M. M. Kumar and N. R. Sunitha, “Hybrid cryptographically secure pseudo-random bit generator,” in *IEEE*, Greater Noida, 2016.

[12] Y. Kurniawan and M. Beta Auditama, “Design and Implementation of Random Number Generator System Based on Android Smartphone Sensor,” in *International Journal of Network Security*, Jawa Barat, 2020.

[13] O. S. Sitompul, F. Rio Naibaho, Z. Situmorang and E. Budhiarti Nababan, “STEGANOGRAPHY WITH HIGHLY RANDOM LINEAR CONGRUENTIAL GENERATOR FOR SECURITY ENHANCEMENT,” in *Third International Conference on Informatics and Computing (ICIC)*, Palembang, 2018.

[14] A. Rock, Pseudorandom Number Generators for Cryptographic Applications, Salzburg: Diplomarbeit zur Erlangung des Magistergrades an der Naturwissenschaftlichen Fakult der Paris-Lodron-Universit¨at Salzburg, 2005.

[15] E. Barker and J. Kelsey, “Recommendation for Random Number Generation Using Deterministic Random Bit Generators,” in *NIST Special Publication 800-90A*, Gaithersburg, 2015.

[16] Python, “What is Python? Executive Summary,” Python, [Online]. Available: https://www.python.org/doc/essays/blurb/. [Accessed 15 July 2023].

[17] pythonbasics, “What is Flask Python,” pythonbasics, [Online]. Available: https://pythonbasics.org/what-is-flask-python/. [Accessed 15 July 2023].

[18] MySQL, “MySQL Workbench: Visual Database Design,” MySQL, [Online]. Available: https://www.mysql.com/products/workbench/design/. [Accessed 15th July 2023].

[19] D. S. Moore, G. P. McCabe and B. A. Craig:, EXCEL MANUAL, Introduction to the Practice of Statistics, New York: W.H. Freeman and Company, 2009.

[20] K. Muralidhar, “Understanding Autocorrelation in Time Series Analysis,” Medium, 4

August 2021. [Online]. Available: https://towardsdatascience.com/understanding-autocorrelation-in-time-series-analysis-322ad52f2199. [Accessed 15 July 2022].

[21] D. E. KNUTH, THE ART OF COMPUTER PROGRAMMING, Stanford: Addison–Wesley, 2014.

[22] L. E. B. III, “Special Publication 800-22 Revision 1a,” in *NIST*, Gaithersburg, 2010.

[23] A. Barth, C. Jackson and J. C. Mitchell, “Robust Defenses for Cross-Site Request Forgery,” in *Stanford University*, 2008.