

Simulating Cryptographic Sensitivity Using Altered Form of Cayley Hashing

Dr.V.Vibitha Kochamani, Assistant Professor,
Department of Mathematics, Mar Dionysius College, Pazhanji, Thrissur, Kerala,
Mail.Id- vibitha@mdcollege.edu.in

Dr. Manjusha K K, Assistant Professor,
Department of Computer Application, Mar Dionysius College, Pazhanji, Thrissur,
Kerala. Mail.Id- manjushakk@mdcollege.edu.in

Abstract

We have defined the modified form of the Cayley hash functions using General linear group. The modified form of the hash function is very efficient and also easy to compute. We studied the Chi-Square Goodness of fit test, which gives the randomness in the output distribution of the hashed value in this defined modified form of hash function which will reduce the chance of occurring collisions and also examined about the Avalanche Effect, where a small change in the input string will gives distinct hashed values, which is the significant feature of a modified form of the hash function.

Keywords: General linear Group, Cayley Hash Function, Girth, Chi-Square (Goodness of fit) test, Avalanche Effect.

Introduction:

Hash functions are simple and easy-to compute, that takes a variable length input and converts it to a fixed-length output [12]. If such a function satisfies additional requirements it can be used for cryptographic applications such as, to protect the authenticity of messages sent over an insecure channel. The basic idea is that the hash result provides a unique imprint of a message, and that the protection of a short imprint is easier than the protection of message itself. A cryptographic hash function can provide assurance of data integrity. Hash functions are widely used in numerous cryptographic protocols and a lot of work has already been put into devising adequate hashing schemes. Hash functions are used as compact representations or digital finger prints of data and to provide message integrity. Some hash functions in current use have been shown to be vulnerable. Early suggestions (particularly SHA family) did

not really use any mathematical ideas apart from Merkle-Damgard [4] construction for producing collision resistant hash functions from collision resistant compression functions, the main idea was just to create a mess by using complex iterations. We have to admit that a mess might be good for hiding purposes, but only to some extent. In the period of time 1990s, Zemor [5] introduced the idea of constructing hash functions from Cayley graphs. Zemor's technique came from a desire to satisfy the following small modification property that introduced in [6].

Proposition 1.1 (Small Modifications Property): There exists a $d \in \mathbb{N}_0$ such that if m' is any modification of m affecting fewer than d consecutive bits then $h(m) \neq h(m')$.

The relation between the small modifications property and hash functions from Cayley graphs is now apparent. Namely, let G be a group with generating set $S = \{A, B\}$ and H be the associated hash function. If the directed girth of $C(G, S)$ is δ , then two messages of length less than δ cannot form a collision in H . From this motivation, Zemor present the idea of using hash functions over the group $SL_2(\mathbb{F}_p)$ with two generators $A = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$ and $B = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$ for a large prime p which was broken by many attacks by their weakness in the factorization. [4] In 1994, Tillich and Zemor's [10] paper proposed a family of hash functions that uses the group of SL_2 over a finite field of 2^n elements as platform for their design. Let n be a positive integer and let $p(x)$ be an irreducible polynomial of degree n over \mathbb{F}_2 . Let A_0 and A_1 be defined as follows: $A_0 = \begin{pmatrix} x & 1 \\ 1 & 0 \end{pmatrix}$ and $A_1 = \begin{pmatrix} x & x+1 \\ 1 & 1 \end{pmatrix}$. Both A_0 and A_1 have determinant 1 over \mathbb{F}_2 . These matrices are the generators of the Tillich-Zemor hash function. Let $m = m_1 m_2 \dots m_k \in \{0, 1\}^*$ be a binary string representation of a message and $K = \mathbb{F}_2[x] / \langle p_n(x) \rangle \cong \mathbb{F}_{2^n}$. The construction of the Tillich-Zemor hash functions also preserves the small modifications property using degree argument in ([8], lemma 3.5).

Lemma 1.1. Suppose that m, m' are bit strings in $\{0, 1\}^*$ such that $H(m) = H(m')$. Then at least one of m, m' must have length greater than n .

In the paper [16,17] we have taken up the following study relevant to the above context. We devised the Hash function as follows: to an arbitrary text of $\{0, 1\}^*$, associate the string of $\{A, B\}$ obtained by substituting 0 for A and 1 for B, then assign to A and B values of adequately chosen matrices of $Heis(\mathbb{Z})$, those could be

$A = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ and $B = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$, then evaluate the product associated with the

string of A and B in the group $Heis(\mathbb{F}_p)$, where \mathbb{F}_p is the field on p elements, p being chosen large prime number and we executed a running time of the hash function using matrix multiplication and also check that the output distributed by hash functions, (that is) hashed values are uniform using the goodness-of-fit test.

Preliminaries

Definition 1: In [9], the vertices ‘x’ of the graph G and one draws a directed edge from x to y, labeled by the group element g for any $x, g \in G$ if and only if $y = xg$. The group consisting of all such vertices and edges will be denoted by Cay(G, G).

Definition 2: The Girth of a graph G, denoted g(G) is the length of the shortest cycle(if any) in G.

Definition 3: The Diameter of a graph G, denoted d(G) is the length of any longest geodesic.

Depending on these requirements Praneel [12] provides the following informal definitions for two different types of hash functions

Definition 4: [12] A One-Way Hash Function is a function h that satisfies the following conditions:

1. The input x can be of arbitrary length and the result h(x) has a fixed length of n bits.
2. Given h and an input x, the computation of h(x) must be easy.
3. The function must be one-way in the sense that given a y in the image of h, it is hard to find a message x such that h(x)= y (pre image-resistance), and given x and h(x) it is hard to find a message $x' \neq x$ such that $h(x') = h(x)$ (second pre-image resistance).

Definition 5: [12] A Collision-Resistant Hash Function is a function h that satisfies the following conditions:

1. The input x can be of arbitrary length and the result h(x) has a fixed length of n bits.
2. Given h and an input x, the computation of h(x) must be easy.
3. The function must be collision-resistant: this means that it is hard to find two distinct messages that hash to the same result (i.e., find x and x' with $x \neq x'$ such that $h(x) = h(x')$), such that $h(x') = h(x)$ (second pre-image resistance).

Definition 6: [12] A Hash Function $h: D \rightarrow R$ where the domain $D = \{0, 1\}^*$, and the range $R = \{0, 1\}^n$ for some $n \geq 1$.

Cayley Hashing

Let $X = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ and $Y = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}$ be the generators of $GL_3(F_p)$ and let $A = X^2, B = Y^2$ and $C = XY$ be the 3×3 matrices of $GL_3(F_p)$. Let $v = m_0 m_1 m_2 \dots m_n$ be any binary string. Then the hash of v is to satisfied the following two conditions as follows:

1. $H(v) = \pi(m_0)\pi(m_1)\pi(m_2)\dots\pi(m_n)$, where $\pi(m_i) = \begin{cases} A & \text{form}_i = 0 \\ B & \text{form}_i = 1 \end{cases}$.

2.If there are three consecutive “one” bits in a row, then the following all “one” bits will be hashed to the matrix C, until we get a three consecutive “zero” bits in a row, as a result hashing the “one” bit is change again to the matrix B.

That is, the bit string 10000111111100011001 will be hashed to the matrix BAAAABBBCCCCAAABBAAB. This hash function is strongly related to the Cayley Graph associated with $GL_3(F_p)$.

Proposition1: The semigroup GL_3 generated by the matrices A,B and C over \mathbb{Z} is free semigroup.

Proof: Let $X = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ and $Y = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}$ be the generators of $GL_3(\mathbb{Z})$ and by

the definition of the free semigroup that “there are no non-trivial relations among the set of generators in a semigroup”. Here, the generators X and Y having no non-trivial relations in semigroup $GL_3(\mathbb{Z})$. Hence, $GL_3(\mathbb{Z})$ form a free semigroup generated by X and Y.

Now, $A = X^2, B = Y^2$ and $C = XY$ be the 3×3 matrices. Thus the generators and their products in free semigroup are all distinct.

Girth of the Cayley Hashing Related to A(k) and B(k):

Let us define $A(k) = \begin{pmatrix} 1 & k & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ and $B(k) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & k \\ 0 & 0 & 1 \end{pmatrix}$. We done the same manner as in [15] as follows:

This is to consider powers of the matrix $C(k)=A(k)B(k)$ will leads to entries larger than p.

Powers of $C(2)=A(2)B(2)$: The matrix of $C(2)$ is $\begin{pmatrix} 1 & 2 & 4 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{pmatrix}$. If

$(C(2))^n = \begin{pmatrix} a_n & b_n & c_n \\ d_n & e_n & f_n \\ g_n & h_n & i_n \end{pmatrix}$, then the recurrence relations as follows:

$$\begin{aligned} a_n &= a_{n-1} + 2d_{n-1} + 4g_{n-1}, & b_n &= b_{n-1} + 2e_{n-1} + 4h_{n-1}, \\ c_n &= c_{n-1} + 2f_{n-1} + 4i_{n-1}, & d_n &= d_{n-1} + 2g_{n-1}, & e_n &= e_{n-1} + 2h_{n-1}, \\ f_n &= f_{n-1} + 2i_{n-1}, & g_n &= g_{n-1}, & h_n &= h_{n-1} \text{ and } i_n = i_{n-1}. \end{aligned}$$

Solving these recurrence relations (with suitable initial conditions) we get,

$$c_n = A \left(\frac{53}{27} + \frac{2\sqrt{78}}{9} \right)^{1/3 n} + B(\alpha + i\beta)^n + c(\alpha - i\beta)^n$$

where, $(\alpha + i\beta)^n = R^n(\cos(n\theta) + i\sin(n\theta))$ and $R = \sqrt{\alpha^2 + \beta^2}, \theta = \tan^{-1} \left(\frac{\alpha}{\beta} \right)$.

Thus c_n is the entry in $(C(2))^n$ and hence conclude that no other entry of $(C(2))^n$ is larger than p as long as $n < \log_{\left(\frac{53}{27} + \frac{2\sqrt{78}}{9}\right)^{1/3}} p$.

The following theorem says that no collision occurs until the length of the bit string is $n < \log_{\left(\frac{53}{27} + \frac{2\sqrt{78}}{9}\right)^{1/3}} p$.

Theorem 1: The girth of the Cayley graph of GL_3 generated by the matrices A(2) and B(2) is greater than $\log_{\left(\frac{53}{27} + \frac{2\sqrt{78}}{9}\right)^{1/3}} p$.

Here the logarithmic base value is $\left(\frac{53}{27} + \frac{2\sqrt{78}}{9}\right)^{1/3} \approx 1.58$. Thus if we choose, $p = 2^{256}$, then there is no collision for the bit strings of length less than 390.

Powers of $C(3)=A(3)B(3)$: The matrix of $C(3)$ is $\begin{pmatrix} 1 & 3 & 9 \\ 0 & 1 & 3 \\ 0 & 0 & 1 \end{pmatrix}$. If

$(C(3))^n = \begin{pmatrix} a_n & b_n & c_n \\ d_n & e_n & f_n \\ g_n & h_n & i_n \end{pmatrix}$, then the recurrence relations as follows:

$$\begin{aligned} a_n &= a_{n-1} + 3d_{n-1} + 9g_{n-1}, & b_n &= b_{n-1} + 3e_{n-1} + 9h_{n-1}, \\ c_n &= c_{n-1} + 3f_{n-1} + 9i_{n-1}, & d_n &= d_{n-1} + 3g_{n-1}, & e_n &= e_{n-1} + 3h_{n-1}, \\ f_n &= f_{n-1} + 3i_{n-1}, & g_n &= g_{n-1}, & h_n &= h_{n-1}, & i_n &= i_{n-1}. \end{aligned}$$

Solving these recurrence relations (with suitable initial conditions) we get,

$$c_n = A \left(\frac{\sqrt{717}}{6} + \frac{241}{54} \right)^{1/3 n} + B(\alpha + i\beta)^n + c(\alpha - i\beta)^n \text{ where } (\alpha \pm i\beta)^n = R^n (\cos(n\theta) \pm i \sin(n\theta)) \text{ and } R = \sqrt{\alpha^2 + \beta^2}, \theta = \tan^{-1} \left(\frac{\alpha}{\beta} \right).$$

Thus c_n is the entry in $(C(3))^n$ and hence conclude that no other entry of $(C(3))^n$ is larger than p as long as $n < \log_{\left(\frac{\sqrt{717}}{6} + \frac{241}{54}\right)^{1/3}} p$.

The following theorem says that no collision occurs until the length of the bit string is $n < \log_{\left(\frac{\sqrt{717}}{6} + \frac{241}{54}\right)^{1/3}} p$.

Theorem2: The girth of the Cayley graph of GL_3 generated by the matrices A(3) and B(3) is greater than $\log_{\left(\frac{\sqrt{717}}{6} + \frac{241}{54}\right)^{1/3}} p$.

Here the logarithmic base value is $\left(\frac{\sqrt{717}}{6} + \frac{241}{54}\right)^{1/3} \approx 2.07$. Thus if we choose, $p = 2^{256}$, then there is no collision for the bit strings of length less than 245.

Remark:

Let us define $A(k) = \begin{pmatrix} 1 & k & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ and $B(k) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & k \\ 0 & 0 & 1 \end{pmatrix}$,

1. In [15], it is proved that the products of n matrices of the form $w(A(k), B(k))$ for integers $k \geq 1$ had almost growth by words $w = (A(k)B(k))^{n/2}$, where n is even.
2. Here, growth of the entries in the matrix of the form $w(A(1), B(1))$ is among all $w(A(k), B(k))$ for $k \geq 1$, and the largest entries in the matrices of the form $(A(1)B(1))^{n/2}$ are $O((1)^n)$.
3. The matrices of the form $w(A(2), B(2))$, its greater matrix element are in $w = (A(2)B(2))^{n/2}$ and its size is $O\left(\left(\left(\frac{53}{27} + \frac{2\sqrt{78}}{9}\right)^{1/3}\right)^n\right)$. That is the girth of the defined modified hash function over the prime field is $O(\log_{1.578} n)$ which is known as the lower bound for the girth in this case. For example, if $p \approx 2^{256}$, then there will not any collisions if both the words having bit strings of length less than 390.
4. Similarly for the matrices of the form $w(A(3), B(3))$, its greater matrix element are in $w = (A(3)B(3))^{n/2}$ and its size is $O\left(\left(\left(\frac{\sqrt{717}}{6} + \frac{241}{54}\right)^{1/3}\right)^n\right)$. That is the girth of the defined modified hash function over the prime field is $O(\log_{2.07} n)$ which is known as the lower bound for the girth in this case. For example, if $p \approx 2^{256}$, then there will not any collisions if both the words having bit strings of length less than 245.

Efficiency

We calculate the average running time to execute the values of hash function $H(m)$ for 100 binary random strings of same length. Here we use matrix multiplication in our observations, we take the pair of generators, $A = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ and $B = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}$

We apply different binary string length of 100, 200, 500, 800, 1000, 2000, 5000. For the efficiency calculation we developed python 3.13 64-bit code. Average running time of the modified Hash function is calculated and displayed in Table 1.

We determined that hash function with small prime ($2^{127} - 1$) to large prime ($2^{256} - 1053$) and it varies from 0.0372 seconds to 0.185 seconds in average of length 100 to 5000.

Avalanche Effect (Security):

Whenever we use the Internet for different purposes like send and receive an email, to do payments, buying products online and so on. Data must be protected from hacking, noise, and interference due to the extensive use of Internet [23]. For proper and secure communication data must be protected. The Avalanche effect or diffusion is a

significant characteristic of a hash function where a minor change in the hash function's input will result in a significantly different output. [22]. The avalanche effect is used for evaluating a cryptographic algorithm's strength [24]. A study conducted by Nazeh et al. (2018) says that the avalanche effect reflected performance of cryptographic algorithm. A good cryptography algorithm has Avalanche effect greater than 50%. [25]

For conducting this experiment, we are using Python 3.13 (64 bit) script as stimulator. The computer has the specifications of 11th Gen Intel(R) Core (TM) i5-1135G7 @ 2.40GHz 2.42 GHz with 8.00 GB and x64-based processor. The input bit strings are provided through text file. This text file is used to read each binary string from the file input. We generate random numbers with the help of a python code and store it to the text file. For Calculating the effect of Avalanche, we are using this concept.

Avalanche Effect Calculation Algorithm:

Input:

- A text file data.txt with **binary input strings** of different length (100,200,...)
- A **mod value** (e.g., 2, 127, 137) provided by the user.

Algorithm Steps:

1. **Matrix Initialization:**
 - Define two constant 3x3 matrices A and B:
2. **Input Processing:**
 - Open and read data.txt.
 - Extract and clean **binary strings**.
3. **Get Modulus:**
 - Prompt the user for a **mod value** (used to wrap matrix multiplication results).
4. **Encrypt Each String:**
 - For each input string:
 - Start with an identity matrix X.
 - For each character in the string:
 - If '0', multiply X by matrix A; if '1', multiply X by matrix B.
 - Store result back into X.
 - After full traversal, apply **modulo operation** on matrix X using the user-provided mod value.
 - Convert resulting matrix into a **binary string**.
5. **Avalanche Effect Calculation:**
 - Convert both encrypted binary strings to integers.
 - Perform XOR to find differing bits.
 - Count the number of '1's in the XOR result (i.e., changed bits).
 - Divide by total number of bits to get the **avalanche effect percentage**.

6. **Output Results:**
- Write intermediate and final results avalanche percentage to result.txt.
 - Display the avalanche percentage on screen.

Result

We computed 100,200,500,800,1000,2000,5000 length bit-strings using Python code and supplied them to the data.txt as file input. The mod values like 1021 (commonly used primes) tend to produce higher avalanche effects around 69.8%, which is generally a good sign in cryptographic functions.

This result shows that even small changes in the input cause significant changes in the output, this makes difficult for an attacker to predict the output based on the input. Our algorithm shows the highest avalanche effect so it is very secure for data communication.

This Figure 1 illustrates how the avalanche effect percentage might vary with different mod value (prime) inputs.

This concept for measuring how sensitive our transformation is to bit flips, which is the essence of the avalanche effect.

VI. Pseudo-Randomness-Chi square Test:

Chi-Square Test:

To get a good hash function we need the distribution of the hashed value is randomly distributed. We know that the elements of prime field are uniformly distributed [20], we examined the output distribution of the hashed values are uniformly distributed by chi-square test.

To examine the chi-square goodness of fit test, the conditions are to be satisfied:

- 1.Choosing the randomly input strings to attain observed frequencies.
- 2.Expected frequencies in every distribution should be greater than or equal to 5.

Using Python code, the Null Hypothesis is tested for the hashed values from the random input strings which are randomly distributed is uniform. The test is conducted for various distribution of the hashed values is performed 100 times and each time with new distribution of hashed values are checked. Hence the hashed values output distribution is uniformly distributed.

VII. Padding: Here, in this defined hash function, we have to add three zeros at the end of the each bit strings to get the hashed value of the original pair (A,B) of 3×3 matrices of $GL_3(F_p)$ due to the conditions in the defined hash functions of step 2.Like this way we can use a minor padding of any bit strings to get hashed value.This useful property preserves the homomorphism of the defined hash function.

III. Comparison of the Cayley Hash Function related to Heisenberg group with our modified form of the hash function

Sl. No	Cayley Hash Function related to Heisenberg Group	Modified Hash Function using General Linear Group
1	Cayley hash function related to Heisenberg group takes the random walks on the Cayley graph using two-generators in $H_{2n}(\mathbb{F}_p)$.	Here, it takes place using three generators, over the general linear group in prime field.
2	The Hashed value of the hash function needs to give $2n$ additions and $3n$ multiplication to hash a bit strings of length n [21]	Here, the hashed value has $(n-1)$ multiplication and $6n$ additions to hash a bit strings of length n .
3	The hash function for large prime $2^{256} - 1053$ without any parallelism hashes to a 10^6 length bit strings in 17.63 seconds in average [21]	Here, in our proposed modified form of hash function for the same large prime $2^{256} - 1053$, hashes to a 10^6 length bit strings in 18.24 seconds in average.
4	Output distribution is randomly generated.	Here, the output of the hash function is distributed randomly by the pseudo-randomness test.

Conclusion

*Here we introduced the new modified hash function with 3-generators 3×3 matrices in general linear group of prime field. This defined hash function is very efficient and easy to compute.

*The entries growth in the matrix of the form $w(A(1), B(1))$ is among all $w(A(k), B(k))$ for $k \geq 1$, and the largest entries in the matrices of the form $(A(1)B(1))^{n/2}$ are $O((1)^n)$.

*The girth of the defined modified hash function over the prime field is $O(\log_{1.578} n)$ which is known as the lower bound for the girth in the case of matrices of the form $w(A(2), B(2))$.

*The girth of the defined modified hash function over the prime field is $O(\log_{2.07} n)$ which is known as the lower bound for the girth in the case of matrices of the form $w(A(3), B(3))$.

*The defined hash functions, output distribution of the hashed value is uniformly random distributed by the chi-square test, it will reduce the collisions in the strings.

*The Avalanche Effect shows that even small changes in the input cause significant changes in the output, this makes difficult for an attacker to predict the output based on the input. Here, it tends to produce higher avalanche effects around 69.8%, which is generally a good sign in hash functions.

References

- [1] Gathmann, A., 2013, "Class Notes on Commutative Algebra," Technische Universität Kaiserslautern.
- [2] Van Rompay, B., "Analysis and Design of Cryptographic Hash Functions, MAC Algorithms and Block Ciphers," Doctoral Dissertation, ISBN: 90-5682-527-5.
- [3] Sosnovski, B., 2016, "Cayley Graphs of Semi-groups and Applications to Hashing," Ph.D. Thesis, City University of New York.
- [4] Stinson, R. D., 1995, "Cryptography Theory and Practice," 2nd Ed., Chapman and Hall/CRC.
- [5] Zemor, G., 1991, "Hash Functions and Graphs with Large Girths," EUROCRYPT (D. W. Davies, Ed.), Lect. Notes Comput. Sci., Springer, LNCS 547, pp. 508–511.
- [6] Godlewski, P., and Camion, P., 1988, "Manipulations and Errors, Detection and Localization," In Advances in Cryptology — EUROCRYPT'88 (Davos Ed.), Lect. Notes Comput. Sci., 330, pp. 97–106.
- [7] Tomkins, H., 2018, "Alternative Generators of the Zemor-Tillich Hash Function: A Quest for Freedom in Projective Linear Groups," University of Ottawa.
- [8] Tillich, J. P., and Zemor, G., 1994, "Hashing with SL_2 ," Advances in Cryptology, Lect. Notes Comput. Sci., Springer-Verlag, 839, pp. 40–49.
- [9] Gallian, J. A., "Contemporary Abstract Algebra," 8th Ed., University of Minnesota, Duluth, ISBN-13: 978-1133599708.
- [10] Dworkin, M. J., 2018, "Permutation-based Hash and Extendable-output Functions," Technical Report.
- [11] Mullan, C., and Tsaban, B., 2016, "SL2 Homomorphic Hash Functions: Worst Case to Average Case Reduction and Short Collision Search," Designs, Codes and Cryptography, 81(1), pp. 83–107.
- [12] Preneel, B., 2003, "Analysis and Design of Cryptographic Hash Functions," Ph.D. Thesis, K.U. Leuven.
- [13] Dang, Q. H., 2015, "Federal Information Processing Standards Publication, Secure Hash Standards," Federal Inf. Process. Stds., 180(4), pp. 33–44.
- [14] Crandall, R., and Pomerance, C., 2005, "Prime Numbers: A Computational Perspective," 2nd Ed., Springer.
- [15] Bromberg, L., Shpilrain, V., and Vdovina, A., 2017, "Navigating in the Cayley Graph of $SL_2(\mathbb{F}_p)$ and Applications to Hashing," Semigroup Forum, 94, pp. 314–324.
- [16] Kochamani, V. V., Lilly, P. L., and Joju, K. T., 2016, "Hashing with Discrete Heisenberg Group and Graph with Large Girth," Int. J. of Theoretical Physics and Cryptography, 5, pp. 1–12.
- [17] Kochamani, V. V., and Lilly, P. L., 2019, "Security Aspects of the Cayley Hash Function Using Discrete Heisenberg Group," J. of Discrete Mathematical Sciences and Cryptography, 5, pp. 1–10.
- [18] Kochamani, V. V., and Lilly, P. L., 2020, "Hash Function Using Free Generators Theorem Over the Projective General Linear Group," Advances in Mathematics: Scientific Journal, 9(4), pp. 2007–2017.
- [19] Kochamani, V. V., and Lilly, P. L., 2025, "Free Generators Theorem in Projective General Linear Group," Communicated.

- [20] Alright, B., 1994, "Essentials of Mathematical Statistics," Jones and Bartlett Publishers.
- [21] Kochamani, V. V., and Lilly, P. L., 2019, "Vectorial Version of the Cayley Hash Function Using Discrete Heisenberg Group," *Malaya Journal of Matematik*, Vol. 5, No. 1, pp. 720–724. <https://doi.org/10.26637/MJM0S01/0128>
- [22] Kumar, A., and Tiwari, N., 2012, "Effective Implementation and Avalanche Effect of AES," *Int. J. of Security, Privacy and Trust Management*, 1(3/4), pp. 31–34.
- [23] Yassein, M. B., Aljawarneh, S., Qawasmeh, E., Mardini, W., and Khamayseh, Y., 2017, "Comprehensive Study of Symmetric Key and Asymmetric Key Encryption Algorithms," 2017 *Int. Conf. on Engineering and Technology (ICET)*, pp. 1–7. IEEE. <https://doi.org/10.1109/ICEngTechnol.2017.8308215>
- [24] Ramanujam, S., and Karuppiah, M., 2011, "Designing an Algorithm with High Avalanche Effect," *IJCSNS Int. J. of Computer Science and Network Security*, 11(1), pp. 106–111. http://paper.ijcsns.org/07_book/201101/20110116.pdf
- [25] Hossain, Md. A., Hossain, Md. B., Imtiaz, S. Md., and Uddin, Md. S., 2016, "Performance Analysis of Different Algorithms," *Int. J. of Advanced Research in Computer Science and Software Engineering*, 6(3), pp. 659–665.