Int. J. Anal. Appl. (2025), 23:305

International Journal of Analysis and Applications

Novel Way of Generating Random Numbers Using Lucas Sequence and Associated in ATM/Ecommerce for Secured Online Transactions

R. Elumalai, G.S.G.N. Anjaneyulu*

Department of Mathematics, SAS, Vellore Institute of Technology, Vellore 632014, India

**Corresponding author:* anjaneyulu.gsgn@vit.ac.in

Abstract. This paper proposes novel pseudorandom number generators (PRNGs) based on the Lucas sequence and the SHA3-512 hashing algorithm over the finite field F_p. We introduce one primary algorithm capable of generating random numbers up to 32 digits (256 bits) in length, suitable for highly confidential applications such as international communications, defense activities, and large monetary transactions. Additionally, three associated sub-algorithms produce fixed-length random numbers of 4, 6, and 8 digits (32, 48, and 64 bits, respectively), optimized for ATM and e-commerce transactions. Unlike existing PRNGs that rely on a single seed, our approach utilizes two seeds: a user-provided, context-specific seed, and a server-generated seed derived from the Lucas sequence over the Pell curve. The server-generated seed remains entirely outside user control, enhancing the security of the generated random numbers. The PRNG process involves hashing the sum of solutions to the Pell curve, converting the resulting hexadecimal hash output into binary, and extracting the first half of the 512 bits, which is then mapped to an integer over the field F_q to produce the random number. Statistical analysis using the Kolmogorov-Smirnov test confirms that the generated numbers follow a uniform distribution. Security analysis demonstrates resilience against various attacks, including direct cryptanalytic, input-based, iterative guessing, backtracking, and gap-filling attacks. These results suggest that the proposed PRNGs offer improved security and efficiency for applications in ATM operations and e-commerce.

1. Introduction

This section delves into the significance of random number generators. Since the late 19th century, researchers have been developing pseudo-random number generators (PRNGs), each claiming to outperform its predecessors. This paper aims to validate such claims and rank existing generators through rigorous empirical testing. PRNGs are categorized into three groups: linear congruential generator-based, linear feedback shift register-based, and cellular automata-based. Representative generators from each category were evaluated using blind statistical tests, the TestU01 library, the NIST statistical test suite, and graphical assessments.

Received: Jun. 5, 2025.

2020 *Mathematics Subject Classification.* 11T71.

Key words and phrases. Cryptoserver; Lucas Sequence; Pell curve; PRNG; SHA-3 512.

https://doi.org/10.28924/2291-8639-23-2025-305
ISSN: 2291-8639

1.1. **Random Number Generator (RNG).** Random number generators (RNGs) are crucial for applications ranging from gaming to cryptography. Cryptographically secure PRNGs (CSPRNGs) must exhibit statistical soundness, attack resistance, and entropy extraction capabilities. Examples include Blum-Blum-Shub and ANSI X9.17. Essential attributes for PRNGs include uniformity, independence, large period, and security, though many fail to satisfy all these requirements.

Small-sized outputs (e.g., 32, 48, or 64 bits) are used in applications like ATMs and online transactions for tasks such as PIN verification and encryption, e.g., in TLS 1.2. High-security applications (e.g., server authentication, session ID generation) require larger random numbers (128 to 512 bits) to prevent brute-force attacks.

1.1.1. *LCG based PRNGs.* Early PRNGs such as von Neumann's middle-square method were replaced by Lehmer's linear congruential generator (LCG), which offered longer periods and better statistical properties

$$x_n = ax_{n-1} \bmod m. \tag{1.1}$$

Lehmer introduced the idea of generating pseudo-random numbers using $u_i = x_i/m$ with carefully selected a and m. An example includes $m = 2^{31} - 1$, $a = 23$, yielding a period of $2^{31} - 2$. This method, tested on IBM's 602A, laid the groundwork for modern PRNGs.

The LCG model was extended by Thomson and Rotenberg (1958):

$$x_n = ax_{n-1} + c \,(\bmod m).$$

Rotenberg showed that with parameters such as $m = 2^{35}$, $a = 2^d + 1$, and odd c, full periods are achievable.

Despite their simplicity and speed, many LCGs fail spectral tests, especially in higher dimensions. RANDU and RANNO were widely used but exhibited poor lattice structure. Marsaglia (1968) and Knuth (1969) analyzed these structures, showing LCG outputs lie on a limited number of hyperplanes, thus encouraging LCG designs with better spectral properties.

To extend periods and improve lattice density, higher-order linear recurrences were introduced:

$$x_n = (a_1 x_{n-1} + ... + a_k x_{n-k}) \bmod m. \tag{1.2}$$

Zierler (1959) and Alanen and Knuth (1964) explored such sequences, with multiple recursive generators (MRGs) emerging as powerful variants.

Nine LCG-based PRNGs studied include Knuth's MMIX LCG (2^{64} period), GNU C Library's rand() ($m = 2^{31}$), lrand48() ($m = 2^{48}$), C++11's minstd_rand ($m = 2^{31} - 1$), Borland LCG ($m = 2^{32}$), MRG31k3p (2^{185} period), and PCG (2^{64} period with permutation output).

1.1.2. *LFSR based PRNGs.* Several LFSR-based PRNGs analyzed include: GNU C Library's random() (BSD-derived, $\approx 16 \times (2^{31} - 1)$ period), Taus88 (3-component, $\approx 2^{88}$ period), LFSR113 ($\approx 2^{113}$), LFSR258 ($\approx 2^{258}$), WELL512a and WELL1024a ($\approx 2^{512} - 1$ and $\approx 2^{1024} - 1$, respectively), and the XORSHIFT family (xorshift32, xorshift64, xorshift1024M_8, and xorshift128+). The MT19937

(Mersenne Twister) and its SIMD-oriented variant SFMT, along with dSFMT optimized for double-precision IEEE 754 outputs, were also included.

1.1.3. *Cellular Automata based PRNGs.* In this study, author proposed several one-dimensional CA-based PRNGs: Rule 30 CA [15], using a 101-cell setup to extract 32-bit values from the center cell states; Hybrid 30-45 CA [16], applying alternating rules with periodic boundaries; Maximal-length CA [16], using null boundaries and site spacing to generate 32-bit outputs; Non-linear 2-state CA [17], a 45-cell non-uniform CA producing 45-bit numbers; and 3-state CA [18], generating ternary strings from a 3-state, 3-neighborhood CA, which are then converted into 32-bit or 64-bit values.

1.2. **Automated Teller Machine (ATM) and E-commerce Applications - Security Aspects.** ATMs and e-commerce systems rely heavily on secure PIN-based transactions. Since their inception from Simjian's early deposit machine in 1960 to Barclays Bank's first ATM in 1967 ATMs have evolved into complex, networked systems supporting global transactions. They use magnetic or chip-based cards and encrypted PINs for secure access. Similarly, e-commerce leverages technologies like mobile payments and encrypted communication to enable secure online transactions. Given the rise in cyber threats targeting financial systems, ensuring the security of PINs (4, 6, or 8 digits) is critical. To address this, we propose three subalgorithms for generating secure, hard-to-predict PINs tailored for ATM and e-commerce use.

1.3. **The Security of ATM Operations using PRNG.** PRNGs are used by ATMs to generate session keys, PINs, and encryption keys, among other things. To stop fraud or illegal access, an ATM's PRNG security is essential. Using a cryptographically secure PRNG guarantees that the generated numbers are unpredictable and resistant to attacks. To protect privacy and stop illegal transactions, ATMs must be equipped with a strong PRNG. The National Institute of Standards and Technology (NIST) has defined a pseudo random number bit generator (PRNG) as a Random Bit Generator (RBG) with a mechanism and entropy input. It produces a sequence of bits from a seed, a secret initial value, and other possible inputs. There are three types of PRNG specified in NIST SP 800-90A: Hash DRBG, HMAC DRBG, and CTR DRBG.

2. Related Works - PRNG Connected in ATM and E-Commerce

The evolution of PRNGs spans from theoretical foundations to modern, high-performance algorithms. Knuth [1] and the GNU C Library [2] established core principles, while Park and Miller [3] and Crawford [4] highlighted the need for reliable generators and refined statistical robustness. L'Ecuyer and collaborators [5], [7], [8], [9] advanced combined recursive and equidistributed generators, greatly improving efficiency and statistical quality. Matsumoto and Nishimura's Mersenne Twister [11] and its SIMD-oriented variants by Saito and Matsumoto [12], [13], [14] became key standards in simulations. O'Neill's PCG family [6] and Vigna's comparative study [10]

further refined performance and evaluation. Ballot and Williams [20] extended theoretical insights through Lucas sequences, linking number theory with random sequence generation. Together, these works mark significant progress toward faster, more reliable, and mathematically grounded PRNGs for scientific computing.

The authors of the papers [19], [21], [22], [23], [24], [25], [26], [27], [28], [29] have explored advancements in cryptographically secure pseudo-random and true random number generators, including ATM authentication mechanisms, QCA-based TRNGs, dual-output ring oscillators, and FPGA-based designs. A common limitation in existing PRNGs is their reliance on a single unique seed, making them vulnerable to prediction.

3. Motivation and Overview of the Research

The drawback of existing PRNGs is that they generate all random numbers from a single, unique seed, which leads to easy prediction of the random number. In 2012, Barker, Elaine, and John Kelsey published NIST SP 800-90a: Recommendation for random number generation using deterministic random bit generators [30]. So the authors are motivated by this drawback to design a new PRNG to overcome the problem by using user-selected seed in the first level and server-selected seed associated with Pell curve solutions to meet the security requirements efficiently to enhance ATM security, which will make it highly difficult to predict the random number. All the functions of ATM will be executed through RNG, like PIN generation, transactions, and OTP-related transactions. In cryptographic applications, output predictability is crucial and it should be intricate. Advanced algorithms are needed to design to meet security criteria.

This paper is organised as follows: Section 4 discusses the mathematical background; Section 5 presents research contributions; Section 6 presents experimental results; and the last section concludes with a summary of our work.

4. Mathematical Work Infrasrtucture: Lucas Sequence over Pell Curve

The Lucas sequence is an integer sequence named after mathematician François 'Edouard Anatole Lucas, who studied it and the Fibonacci sequence. Lucas numbers are individual numbers in the sequence, which form complementary instances of Lucas sequences. The sequence has a recursive relationship with the Fibonacci sequence, where each term is the sum of two previous terms with different starting values. The terms are rounding's of integer powers of the golden ratio.

4.1. Algebraic Structure of Lucas Sequence.

In this subsection, we will derive Lucas sequences $x_i(m,k)$ and $y_i(m,k)$ that satisfies the Pell curve $P_{N,4k^i} : x_i^2(m,k) - Ny_i^2(m,k) = 4k^i$. In mathematics, the Lucas sequences $x_i(m,k)$ and $y_i(m,k)$ are certain constant-recursive integer sequences that satisfy the recurrence relation

$$s_i = m.s_{i-1} - k.s_{i-2}. \tag{4.1}$$

Hence we have,

$$x_i = m.x_{i-1} - k.x_{i-2} \tag{4.2}$$

$$y_i = m.y_{i-1} - k.y_{i-2} \tag{4.3}$$

where m and k are fixed integers. Any sequence satisfying this recurrence relation can be represented as a linear combination of the Lucas sequences $x_i(m,k)$ and $y_i(m,k)$. More generally, Lucas sequences $x_i(m,k)$ and $y_i(m,k)$ represent sequences of polynomials in m and k with integer coefficients.

A monic polynomial of degree two with coefficients m and k can be represented as

$$x^2 - mx + k = 0. \tag{4.4}$$

There are two real-number solutions α and β of the above equation are as follows:

$$\alpha = \frac{m + \sqrt{N}}{2}, \text{ where } N = m^2 - 4k \tag{4.5}$$

$$\beta = \frac{m - \sqrt{N}}{2}. \tag{4.6}$$

Then from equations (4.2), (4.3), (4.5) and (4.6), we have

$$\alpha^i = \frac{x_i(m,k) + y_i(m,k)\sqrt{N}}{2} \tag{4.7}$$

$$\beta^i = \frac{x_i(m,k) - y_i(m,k)\sqrt{N}}{2}. \tag{4.8}$$

Then adding and subtracting equation (4.7) and (4.8), we have

$$x_i(m,k) = \alpha^i + \beta^i \tag{4.9}$$

$$y_i(m,k) = \frac{\alpha^i - \beta^i}{\alpha - \beta} = \frac{\alpha^i - \beta^i}{\sqrt{N}}. \tag{4.10}$$

Then squaring equation (4.9) and (4.10), we have

$$x_i(m,k)^2 - Ny_i(m,k)^2 = 4k^i. \tag{4.11}$$

Equation (4.11) is known as Pell curve and it is denoted by $P_{N,4k^i}$. The Lucas sequences $x_i(m,k)$ and $y_i(m,k)$ can be obtained from equations (4.2) and (4.3). The ordered pair of these sequences is a solution of the Pell curve $P_{N,4k^i}$, with a specific curve for each n value. The following iteration may be used to determine the first solution of each Pell curve $P_{N,4k^i}$:

$$x_0(m,k) = 2 \text{ and } x_1(m,k) = m \tag{4.12}$$

$$y_0(m,k) = 0 \text{ and } y_1(m,k) = 1 \tag{4.13}$$

$$x_i(m,k) = m.x_{i-1}(m,k) - k.x_{i-2}(m,k) \tag{4.14}$$

$$y_i(m,k) = m.y_{i-1}(m,k) - k.y_{i-2}(m,k). \tag{4.15}$$

The Lucas sequence, including Fibonacci, Lucas, Pell, Jacobsthal, Mersenne, and Fermat numbers, is a special case of the sequence for certain values of m and k, which are depicted in FIG. 1 as follows:

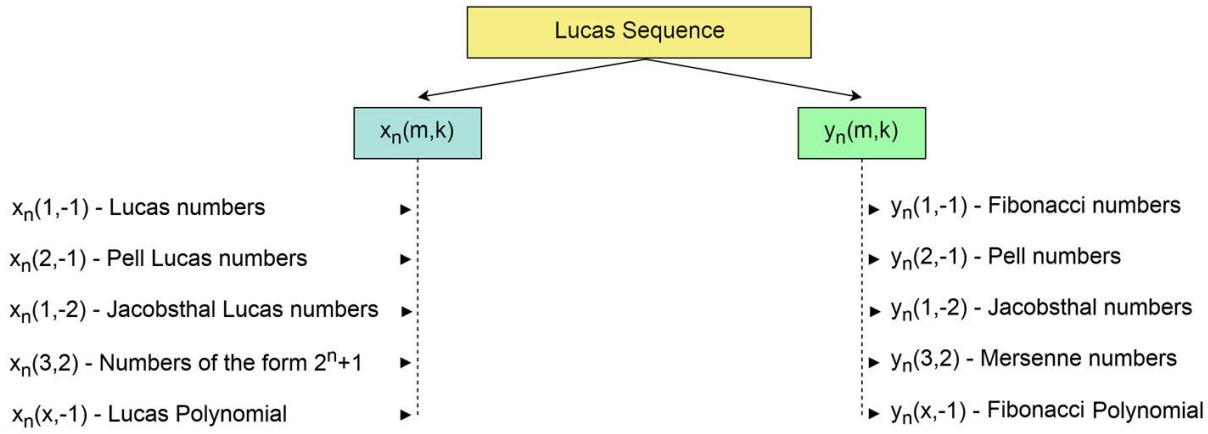


FIG. 1. The above sequences of pair $(x_n(1,-1), y_n(1,-1))$, $(x_n(2,-1), y_n(2,-1))$, $(x_n(1,-2), y_n(1,-2))$, $(x_n(3,2), y_n(3,2))$ are solutions of the coressponding Pell curve equation $P_{5,4(-1)^n}$, $P_{8,4(-1)^n}$, $P_{9,4(-2)^n}$ and $P_{1,4(2)^n}$.

The Lucas sequence is a sequence that serves as the solution to the Pell curve. TABLE 1, columns 2 and 3, show that there are infinitely many Pell curves, each with infinite solutions. If the RHS value in a Pell curve is square, such as $4k^{2i} = 4, 4k^2, 4k^4, ...$, there are infinitely many rational solutions as per the Brahmagupta identity. As $i \to \infty$ means the number of iterations becomes large and coresspondingly a Pell curve will be of the form $x^2 - Ny^2 = 4k^{\infty}$. So the initial solution to this Pell curve will also be large, as can be seen in TABLE 1, column 3. TABLE 1 displays a series of polynomials that may be any of the beginning values of m and k for the first random integer. The values of n, m, and k for the second random number do not fulfil the Pell equation $P_{N,4k^n}$.

n value	Pell Curve	Initial Solution
0	$x^2 - Ny^2 = 4$	(2,0)
1	$x^2 - Ny^2 = 4k$	(m,1)
2	$x^2 - Ny^2 = 4k^2$	$(m^2 - 2k, m)$
3	$x^2 - Ny^2 = 4k^3$	$(m^3 - 2mk, m^2 - k)$
4	$x^2 - Ny^2 = 4k^4$	$(m^4 - 3m^2k + 2k^2, m^3 - 2mk)$
5	$x^2 - Ny^2 = 4k^5$	$(m^5 - 4m^3k + 4mk^2, m^4 - 3m^2k + k^2)$

TABLE 1. There are an infinite number of Pell curves of the form $x^2 - Ny^2 = 4k^n$; we have shown only a few among them with their initial solution.

5. Research Contributions: New Projections on Pell Curve

In our contributions, we are proposing a main algorithm that can produce random numbers of any number of digits. In general, a huge random number of digits will be useful to secure information from intruders in international affairs, space organisations, and defence-related activities of any country. In specific, the random numbers of small numbers of digits will be particularly applicable for ATM-related transactions due to their lower computational capacity. So we propose three additional algorithms in Section 5.3, maintaining the same level of hardness while reducing the complexity of the main algorithm to improve the speed of generating random numbers, as shown in FIG. 2.

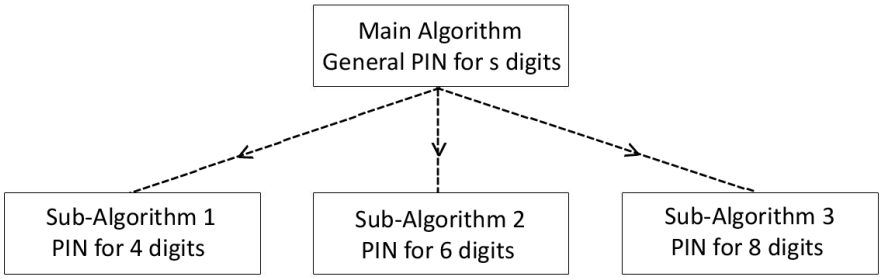


Fig. 2. Main Algorithm and Sub-Algorithms

In this section, we focus on the infinite solutions of the Pell equation. The solutions of the Pell equation can be reduced from infinite to finite by considering them over a finite field. Both sets of solutions (infinite solutions over the reals and finite solutions over a finite field) are symmetric about the x-axis. Knowing at least one solution to the Pell equation allows us to generate infinitely many more solutions; the initial solution is referred to as the *fundamental solution*.

5.1. **Initial Setup.** The proposed algorithm is based on the Pell curve, the SHA3-512 hash algorithm, and finite field arithmetic. When the value of i increases, the correlation between x_i and y_i diverges rapidly. To improve the efficiency of the algorithm, we apply $\bmod p$ to S_i. The output R_i depends on user requirements.

Next, we define the required parameters S_i, B_i, I_i, and R_i, which are used in the proposed algorithm along with the Lucas sequences $x_i(m,k)$ and $y_i(m,k)$. Hereafter, for simplicity, we denote $x_i(m,k)$ and $y_i(m,k)$ as x_i and y_i, respectively.

5.2. **Basic Architecture for Main Algorithm.**

PRNG of Fixed Length using Lucas Sequence with Sequential Iterations

Step 1: The PRNG accepts an input, $m = s_2$ bit, $k = (2s_2 - \epsilon_2)$ bit, $\epsilon_2 \geq 4$ and i.

Step 2: The PRNG maintains an ever-changing state using Lucas sequence. i.e. $x_i = mx_{i-1} - kx_{i-2}$ and $y_i = my_{i-1} - ky_{i-2}$.

Step 3: The PRNG accepts an optional input, a and c.
This may be assumed to be 1 and 0 if not suppiled.

Step 4: The PRNG accepts input prime number p and q.

Step 5: The PRNG generates each output as follows:
(a) $S_i = a(x_i + y_i) + c \ (\text{mod } p)$
(b) $B_i = \text{hash}(S_i)$
(c) $I'_i =$ Left 256 bit of B_i
(d) $R_i = I'_i \ (\text{mod } q)$

Step 6: Set $i = i + 1$, sequential iterations.

Initial values of m, k can be any values derived from the monic polynomial of degree two $x^2 - mx + k = 0$, with i being a suitable iterated number $(i \geq 2)$. However, in the next iteration, the values m, k, and i are obtained from the previous output of SHA-3 512, so these values do not satisfy the Pell equation $P_{N,4k^i}$. This process of generating random number is shown in FIG. 3.

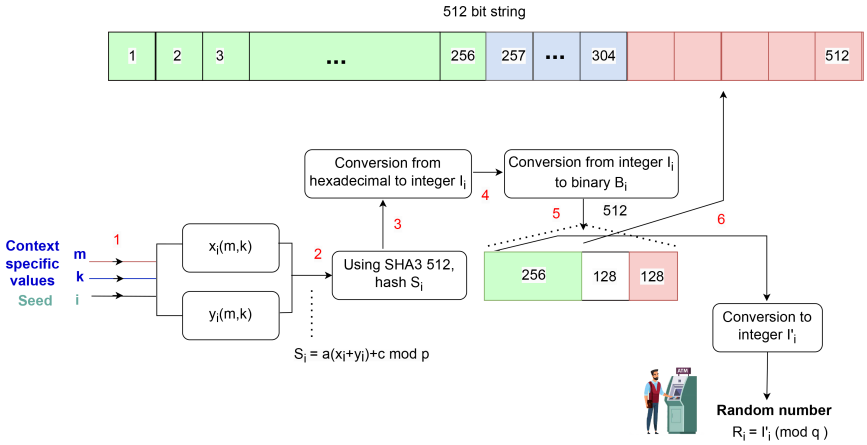


FIG. 3. Main algorithm to generate pseudorandom number for huge number of digits.

The PRNG with a fixed output length can be described in pseudocode in Algorithm 5.1 as follows:

Algorithm 5.1 PRNG with a fixed output length

Require: (m, k, i, p, q)

Ensure: Random number (256 bit) R_i

1: Select $m, k, i \in \mathbb{N}$ (i.e. $m^2 > 4k$) Initial Seed
2: Select prime p & q
3: Select secret key N
4: $x_i(m,k) \leftarrow m.x_{i-1}(m,k) - k.x_{i-2}(m,k)$ Lucas sequence
5: $y_i(m,k) \leftarrow m.y_{i-1}(m,k) - k.y_{i-2}(m,k)$ Lucas sequence
6: $S_i \leftarrow a(x_i(m,k) + y_i(m,k)) + c \bmod \mathrm{p}$
7: $I_i \leftarrow \text{hashing}(S_i)$ using SHA-3 512 Conversion to integer
8: $B_i \leftarrow$ Integer I_i Conversion to binary
9: $I_i' \leftarrow$ First 256 bit of B_i Conversion to integer
10: $R_i \leftarrow I_i' (\text{mod } q)$ Pseudo Random Number
11: **repeat**
12: $\quad i = i + 1$
13: **until** R_i is obtained
14: **return** Random number R_i

The three initial parameters m, k, and i are kept secret, and the value of i changes in each iteration. The number of iterations is also kept confidential, making it challenging to predict the next random number due to their confidentiality. Lucas sequences x_i and y_i are solutions of the corresponding Pell curve $x_i^2 - Ny_i^2 = 4k^i$. The random numbers R_i are generated after i being increased by one, say $i + 1$. i.e. If R_i is the initial random number, then R_{i+1} is the subsequent random number.

Remark 5.1. *Although SHA-3 512 often produces output with 512 bits, there are instances in which it produces output with one, two, or even three bits less than 512. If we assign a right of 512 after splitting 512 into two 256-bit segments, there could not be a guarantee that the output would include 256 bits. Thus, we have decided to choose the left half of 256. The time required to create the output value increases when the n value is more than 8 bits. We limit the three variables to 32 bits for m, 16 bits for k, and 8 bits for n in order to maximize performance.*

5.3. **Sub-Algorithms: For Low Computational Capacity Devices.** Since in day-to-day applications, an enormous number of random numbers are generated in lakes, it is highly difficult to choose a value that is less than x_i and y_i as an input to the system by the user in each iteration. Therefore, to make the procedure automated and to improve the hardness of the random number, we chose this value from the Pell curve solution x_i, y_i as $x_i + y_i$. All three algorithms are explained below consecutively. ATM systems require lower computational capacity, such as 32 bit, 48 bit, or 64 bit, depending on the specific system requirements. Thus the proposed algorithms are implemented in ATM.

5.3.1. *Algorithm for generating pseudorandom numbers of 4, 6 and 8 digits.* In Step 10 of the main algorithm, suppose we choose the value of q as 10^5, 10^7, and 10^9; then 4-digit, 6-digit, and 8-digit random numbers are generated, respectively. The corresponding flowchart for this process is depicted in FIG. 4.

For example, to generate an Indian ATM or debit card password, a 4-digit PIN is required. However, P_n consists of 64 digits. To reduce 64 digits to 4 digits, we apply $(\mathrm{mod})\, q$ to P_n; in this case, $(\mathrm{mod})\, 10^5$.

Consider another scenario, such as an SBI online transaction, which uses an 8-digit PIN. To generate this, we apply $(\mathrm{mod})\, 10^9$ to P_n to obtain an 8-digit PIN. Thus, the value of q depends on the user requirement.

The PRNG algorithm uses Lucas sequence and SHA-3 hashing algorithm to generate random numbers from three input values. Initial value of m, k and i can be any of positive integers such that $m > k$ to produce the first random number. But, in the next iteration, the values m, k and i are obtained from previous output of SHA-3 512, so these values does not satisfy the Pell equation $P_{N,4k^i}$. Repeating this process i times yields i random numbers. The bit size of a random number R_i is determined by the user's requirement based on which the prime number q is chosen. In this case, q is chosen to be 512 bit. There are many steps involved in the process of producing a random number, as shown in FIG. 4, which are thoroughly explained in Algorithm 5.1. The path 1–2–3–4–5–(6) results in a random number R_i generated at step 6 (where (6) indicates the random number generation step). The process is as follows: Step 1: Seed initialization based on a Pell curve solution. Step 2: Compute the sum of Pell curve solutions over a finite field $\mathbb{F}_p$. Step 3: Hash the result obtained from step 2 over $\mathbb{F}_p$. Step 4: Convert the hash value from hexadecimal to integer. Step 5: Convert the resulting integer into binary form. Step 6: Generate a random number R_i from the binary representation. Final Step: Update the initial seed i with the new seed $i+1$ (incremented by one). The proposed PRNG is useful for cryptoservers and also in the Monte Carlo method, electronic games, machine learning, and deep learning. The proposed PRNG is equivalent to BBS if the prime $q = 2$ can find implementations on platforms like GitHub and Maple.

6. Experimental Results

The results were computed using Java version 19.0.1, and an HP 240 G5 Notebook PC with a 1900 MHz CPU and 4019 MB RAM.

6.1. **Output of the proposed PRNG for 4, 6, and 8 digits: First 1000 Iterations.** The results are shown in TABLES 2, 3 and 4.

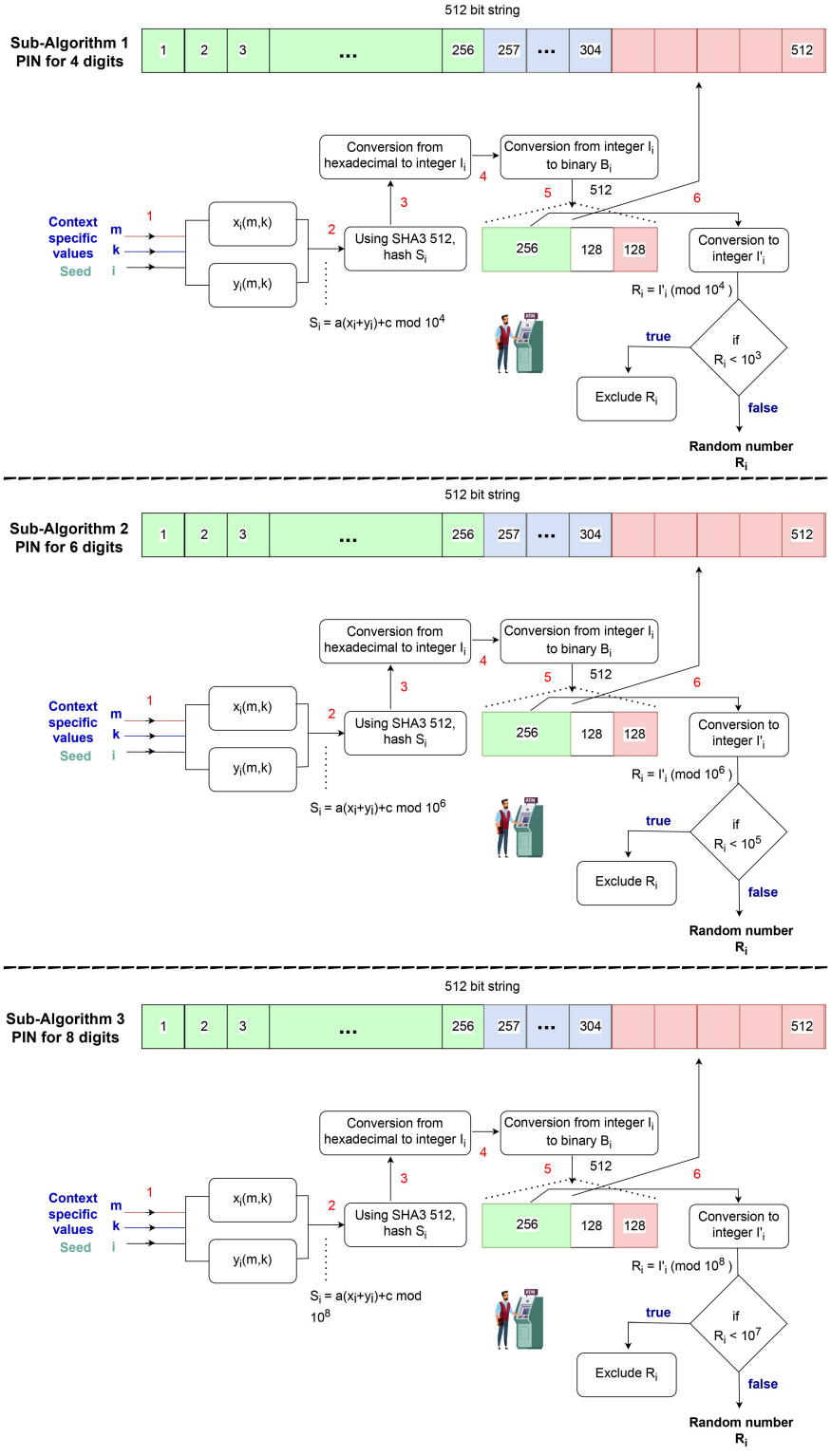


FIG. 4. Sub-algorithms to generate pseudorandom numbers of 4 digits, 6 digits, and 8 digits for ATM and online transactions.

6.1.1. *Results of Sub-Algorithm 1: Random Numbers for 4 digit.*

$x_0 = 2; x_1 = m; y_0 = 0; y_1 = 1; m = 2; k = 3; a = 23; c = 0; p, q = 3311169839$

3189	7748	1554	4158	2637	7187	8132	6852	1350	3173	1801	8062	4410	5458	2529	1433	8387
3984	3121	4712	7641	9152	7312	5591	3128	3145	6004	2990	2244	1371	7660	4955	5817	8063
2512	9256	8901	2559	7795	3206	1168	5898	8983	9886	8270	1504	8093	3174	3989	9077	1662
8179	8242	3200	1647	1973	8155	5817	4870	3166	3597	3820	1086	7873	8839	1072	7100	9632
1096	7347	9800	9677	4127	2088	7080	7473	8809	4158	8665	4963	7391	1366	9472	1051	3141
2364	8230	6359	3495	8692	3694	8572	2376	7636	8589	1705	4343	8179	8571	8516	7770	1212
4440	4353	9582	2686	7953	4667	9031	1402	9748	2837	2528	4106	8016	8850	9654	8887	3897
2481	8279	3384	5840	6115	1822	7550	7534	7893	7763	2833	3269	8556	1935	6472	3329	2015
3750	1016	5986	4629	6486	8591	2519	3365	8569	2162	7829	1080	2775	1567	9067	9862	4400
4402	5309	2662	8357	3805	6238	2948	8587	3285	6843	4768	4968	7840	4712	8387	5976	2359
8662	2688	7561	9812	5362	8930	4401	3836	2730	4585	8787	6370	3218	4122	2884	3354	7254
9384	4450	5565	1542	7333	4119	4773	1927	1191	6493	7459	1962	3126	8198	3364	2139	8850
5184	6770	8743	3781	4649	7334	3218	4739	4661	3473	1992	1642	2124	3982	3766	3150	1763
3230	9693	2227	8572	1889	6576	2374	1913	8751	1590	5061	8761	6773	4119	9886	1106	3969
2206	3121	2681	8627	5631	9279	1358	8567	1768	6740	3779	4423	9926	2604	6628	7047	7301
3034	5015	4270	2250	1460	5967	1732	4881	8713	8673	1504	7475	4880	1672	2419	5924	5095
4331	5102	9375	6821	2936	5239	8755	3717	5937	9090	2904	6043	9923	8900	3027	5933	8242
8329	7548	4680	9968	7369	7080	5749	2827	3154	8921	7713	7534	8900	3789	6805	4647	7238
3538	8544	8367	5335	1554	5324	2521	3561	6686	6036	7595	3315	1825	9899	4353	7989	7031
8740	2772	3657	5025	8937	6549	5788	6400	1542	4363	9791	5824	7361	9926	1692	1605	7891
6715	5501	7391	1743	7547	1238	8520	3269	2194	3160	8227	6905	5352	2658	3332	6779	2545
8751	4204	4542	5761	9691	5661	3908	2964	2541	7556	2785	3657	6110	3986	7893	3576	6387
6238	6830	2534	1136	1657	8791	5208	8955	6198	6009	1967	9514	5190	1379	8051	8553	5070
5950	4142	8132	9144	1997	7592	8930	8798	6773	5639	8902	3492	3366	9384	4992	2959	2818
1257	1113	5458	2655	5745	7997	3095	7437	2270	2231	6942	1038	4267	5522	7186	9110	9168
6463	6376	8250	9335	2788	6455	5016	3145	3473	7956	5854	9055	8293	5967	1387	7989	4312
5455	6567	7829	8024	3130	6762	4393	8591	2620	3203	8224	8337	7873	2947	7234	5548	7028
3121	4924	8601	1003	6845	8992	9873	6215	6366	3699	9400	7681	8279	8087	5324	5997	5190
3165	3837	8242	4483	3766	9774	8022	8930	1131	1504	8155	6206	2709	4323	3790	6477	6359
6961	9240	7633	7475	3908	4327	2180	3384	8856	3296	3682	3130	1298	2603	1537	4465	9873
4470	4713	7413	1566	9556	7080	6820	2706	6873	7681	8639	5559	2573	2016	3237	6253	2868
1902	3821	5352	4794	9252	2335	2469	1058	5607	7563	9800	8407	9407	4353	7296	5465	4955
3797	2887	4402	8589	2194	3036	7867	6882	5653	3981	1346	5024	4705	8602	7660	5661	2317
3350	3873	4285	7919	3389	1050	9344	4564	3269	2508	7387	7651	9161	9582	8798	2088	6212
3970	1483	1759	6852	4999	3996	6510	1889	2936	2863	4155	7246	3205	1402	8154	3135	9200
3976	9585	6238	8009	5672	6890	4476	8629	9770	3698	2612	4925	9240	7301	8856	1434	1798
4443	3157	2868	1475	2564	9053	2261	2658	5518	6812	3766	4088	9384	8779	5997	8099	6842
7128	4646	1917	2472	3682	6949	9200	4944	2715	3912	7713	6787	2878	5032	1444	8629	4056
5345	6791	6858	8001	3473	2001	5521	4204	1555	3905	1168	3192	3488	4871	6090	2475	3154
7699	2165	7830	5940	7199	3889	3388	5107	3218	2872	8181	2832	1611	2392	3121	8518	6297
1844	4587	5825	8242	3085	4540	9454	6929	7612	6060	2415	1791	8198	1247	8662	1038	4836
5352	5441	2772	5250	9115	1504	4349	5566	6798	7681	1475	1058	2046	7148	5908	3181	7337
1080	1874	6586	2976	6097	2270	5977	3218	2573	5672	1358	4655	5800	9564	4657	8007	7080
5541	5235	2138	5311	7186	9104	6387	8659	4992	6854	4142	1524	3049	5497	8152	8227	5910
5009	2287	7338	6477	9537	8790	7550	7849	5781	4353	3230	8661	2415	3174	4857	6315	2335
6851	7435	2500	7129	9879	7244	6621	4419	1000	2463	5781	9104	2787	2062	8920	9114	3269
1819	6536	6223	1433	1648	6370	4578	2440	5675	3714	1094	6801	2407	1770	4963	6947	8913
5817	9169	3865	7393	6456	5307	9289	4967	8583	2573	6238	4304	2235	1612	8250	3583	5670
4368	3073	4627	6164	6317	5756	6808	4647	4644	6632	3682	3750	1298	9093	1633	9384	9004
4948	8290	2540	2872	1190	6525	4034	2863	2904	3347	4034	9672	4534	8377	5653	9477	3162
4822	2902	3858	5208	3351	3429	7455	8936	3388	3473	5840	4283	7634	6275	3013	5526	1320
4090	3322	8502	3230	9252	5143	3346	7057	9082	1734	6801	8553	1485	8001	3606	1803	7848
3351	1891	3121	8911	6928	2139	6743	1250	7347	9277	6471	5697	3494	4419	9808	6702	9342
9050	9410															

TABLE 2. There are 1000 iterations in the iteration process; 87 integers with less than four digits are eliminated, and the remaining 903 numbers are four-digit random numbers R_i which are shown row-wise above for Sub-Algorithm 1.

6.1.2. *Results of Sub-Algorithm 2: Random Numbers for 6 digit.*

$x_0 = 2; x_1 = m; y_0 = 0; y_1 = 1; m = 2; k = 3; a = 23; c = 0; p, q = 3311169839$

593189	521900	336696	521900	741554	624158	200363	369950	133033	921146	336852	335611	881583	833447
277757	219794	904717	180485	501567	625353	914408	217895	771858	834833	935939	971602	821044	903911
702313	316101	524682	340136	226589	443216	604459	773518	633517	535623	910008	837530	729101	418989
444310	648787	796336	963917	985969	507707	260422	715413	965124	144016	672432	849240	325690	806976
460359	613218	152139	312740	472333	495145	461054	498199	659876	292555	507211	455708	650896	652936
999435	910064	774474	519666	843682	992460	106836	359688	343947	522325	960752	560848	589157	178766
694032	895571	572735	815607	969175	549197	597187	508585	444123	269065	638249	277208	241133	488153
225739	209182	807460	531418	422278	688332	315488	250465	942671	777273	882567	345311	357862	857296
793017	370502	707412	394317	182867	677808	294207	800490	828083	471023	614144	470506	458654	843864
635219	474614	652056	947045	527671	328024	422129	874698	919516	993811	321287	384184	147393	593660
452229	976888	290359	536979	647212	907157	701360	360736	172876	935592	599083	123733	548320	794606
503433	895784	375576	650756	246100	249406	436070	634748	341125	902934	526139	787032	691773	661575
915795	488064	141490	532952	300309	204699	690795	542036	208666	316143	375497	215148	291670	125600
846514	660818	528916	232605	905530	758315	101131	414046	763059	342778	931710	658827	322455	215345
201143	784877	360368	695669	500538	131816	912655	389383	995610	176020	167548	999511	479295	220527
845469	477579	314027	576017	893300	844289	665639	362129	175464	411455	111231	445061	723184	770456
919347	506159	113379	987290	197388	444178	571950	866423	592636	455061	572823	172656	336877	913850
949207	624921	827027	989924	644658	791037	146093	686629	669885	263975	722689	217150	565505	842354
646945	718613	770649	763026	932511	247653	670919	994573	786150	590196	969030	736630	776988	266452
418600	454357	198285	283488	167793	705549	950116	659066	894283	244830	298756	826906	705332	579143
281608	137611	219936	287607	508356	805558	736375	843027	244140	374325	265903	653946	741375	425472
285675	322091	926448	388971	564257	633671	283151	925045	116584	657747	317236	868481	998485	549837
893940	103182	708747	197776	682521	997499	671453	660224	260993	490952	158712	160873	453447	780151
746001	416462	656961	748766	232725	251549	820233	919970	615791	529046	145156	441910	354790	763343
339196	291551	112525	546084	569304	430461	428081	845667	400194	110810	487306	475799	733066	457135
685476	892756	157522	776437	926183	337273	803316	560474	895127	689008	506139	250812	177986	725678
975704	756390	796844	942505	135493	794556	374984	138242	253868	498196	237157	414674	581282	958029
939720	717258	719637	815266	944308	306683	720589	623668	460968	828511	974620	636336	250088	612942
122864	710117	983528	199095	470981	257920	900356	425961	691976	907588	991347	567464	748727	131532
621014	448647	291807	659930	213458	564144	313178	708974	247668	957803	452557	959594	270903	352609
584252	323183	336968	281699	659284	979144	423158	574874	705055	596006	631457	736630	465887	476132
381601	848244	272300	618230	526239	910666	632816	362308	391034	949056	123400	742781	534451	653165
293945	938825	281351	108795	805503	286398	709167	894918	633367	689965	807149	152683	352673	274536
490137	687561	876217	582192	174844	645618	384430	660378	152225	656678	457359	241851	473623	170084
175002	500158	589604	878469	185743	788749	900301	108666	205151	402587	249094	319676	250781	923386
600563	621802	723441	447637	588770	621994	424370	115141	342804	125082	687294	634191	654616	770558
954665	543686	411707	366826	146267	428697	875407	509267	783602	824353	386017	113046	258457	465466
157994	243272	400749	321851	157468	741504	893622	371200	116300	394182	120187	450164	522887	301776
593088	237768	519516	397323	820408	929492	231468	144123	445436	767300	977987	512941	545163	638943
103683	503163	285422	126343	265833	301466	737967	539279	413873	180781	693161	227787	358076	532537
182934	798347	825189	454236	314010	697887	937204	225333	732925	423179	330373	862784	423522	465009
328115	517676	549415	463819	190900	259972	922297	306247	259700	482683	203010	976955	549822	941707
933184	564263	210486	558338	644106	881488	931241	405116	170645	821549	312910	597820	873941	702261
119149	385419	675091	465879	336516	603762	487306	173371	993958	783116	501830	271521	886504	887929
876731	790786	482680	703206	880613	765200	398763	396542	918883	823155	729026	372144	459902	396399
715625	399414	851559	134514	414339	976002	962985	506239	282557	280770	261457	114573	680114	658131
490887	993143	760604	697431	905641	297355	982785	910874	202359	811229	481892	837424	260496	257605
449210	621268	915696	499832	247203	478217	524598	840091	154053	238663	164079	784817	304840	171363
657128	781617	195641	586844	795613	658611	197193	877907	694969	783471	818262	617591	819255	455426
354154	183551	446466	926907	269568	261934	566071	187008	663439	743063	918505	761087	524207	109545
681091	573374	759509	500340	492202	538247	134600	276497	249544	709814	462435	326424	689487	108176
815210	725361	498401	124871	216549	163688	938143	422383	458991	736450	180025	247672	601060	618556
221792	557778	485187	840416	523158	275961	254739	581518	413934	199260	775081	610167	927413	148846
911665	538765	997873	754183	613181	878846	869652	523590	473987	951951	771509	395743	161218	379833
227975	584511	385830	931295	483822	146349	573012	145350	949458	763015	564094	882828	479313	623911
252440	171188	384702	133631	211640	703229	119295	434598	370655	692752	595209	675985	740756	122461
712461	110084	269229	657512	260753	640605	183202	677054	718887	258764	351380	549789	173646	589895
986456	744784	522151	382800	198792	620177	804571	374487	811659	210998	381381	646175	338707	771638
397237	707254	424573	566708	407156	417503	367048	853655	841301	723034	864991	390534	859765	186861
261013	866398	989287	716031	416225	803452	787792	185861	987619	403483	841477	881540	211938	770948
332745	781689	406528	772180	417766	545459	149269	687118	581064	141109	133345	905140	712491	729360
476714	971709	417333	526185	220601	755688	287756	133807	911659	946312	447478	355634	837420	978028
678845	696808	661983	148668	981119	711135	535652	725767	302083	669180	706883	995663	780562	449481
913126	830923	225867	136479	434546	467781	721117	725924	824032	201008	516650	950163	245933	270360
265554	579699	140761	208009	953759	632014	968148	198446	735996	772278	652839			

TABLE 3. There are 1000 iterations in the iteration process; 83 integers with less than six digits are eliminated, and the remaining 907 numbers are six-digit random numbers R_i which are shown row-wise above for Algorithm 2.

6.1.3. *Results of Sub-Algorithm 3: Random Numbers for 4,6 and 8 digit.*

$x_0 = 2; x_1 = m; y_0 = 0; y_1 = 1; m = 2; k = 3; a = 23; c = 0; p, q = 3311169839$

14593189	42446047	80741554	34624158	47200363	56194735	85122716	87404632	62336852	57335611	51881583
77176704	20459981	32163511	77180485	83440171	52249689	20377112	19781731	16069044	21712388	82979651
92821044	34237323	71266304	13574399	80379352	53168712	21645044	44663263	44172555	62197704	67240581
26357174	83884532	79815798	62824691	85564632	92755610	39100102	28666615	96187842	81336080	28752872
82882201	88760320	23888266	60629235	11342320	73379574	13319834	70469555	36268945	83480208	35461054
81468914	82024903	67944298	15705731	40728868	72795442	47311384	64245649	35750815	68524067	49638060
98166150	54844994	97098744	92611391	71009113	80210146	60247988	39153537	77234676	39013774	25426442
12474034	71654716	27097826	31420299	42386327	30714092	80929148	27611837	34109614	94119466	10456070
46565557	89749227	62096910	61272376	16884111	49952772	22481651	51783868	59588882	72060349	47942671
61455102	95772100	58116412	78045086	80339445	20619256	84340223	72791916	61506532	96512796	14202871
67862139	74958510	90426702	98245945	96043818	80434123	37365396	19516996	44428133	53069456	25283167
79699674	39652056	84301210	67079494	81669294	36380961	37849149	13217209	33751923	30178998	95363561
18097636	89411116	61286582	42965356	48145277	30786943	23187002	56473970	56829752	89741210	42438328
10122477	52453650	98976472	53028569	78042734	38798070	13745812	55474860	43055801	52533060	66125581
30737119	41956215	84955623	95488491	38559951	35420917	36152320	14614725	18243701	91408706	22162248
69922741	87190186	71052405	63736847	63908119	35696977	42167495	75417730	78848617	82958149	32336704
67299384	71564078	93104904	91298533	55866812	97194430	81925000	85696970	98889351	94013845	44799098
19503940	84907955	97957008	92173544	30803778	65780922	59733869	61201996	58011528	82814868	81420102
66514658	10820803	94063419	26259279	15499058	67902131	40514727	61491939	62202187	17512818	91973831
32297325	41042298	60984395	58214364	11117060	38421390	74360126	54783763	94496470	90805860	79005978
17725885	24009706	46050487	97737965	51912958	82196155	50497731	56276295	31322355	39394945	95372720
41794524	30617391	89367511	64644015	68217150	34504945	54550105	30076471	42379045	85930411	52813625
18767572	75785313	12862812	61478177	85229407	60482063	52820845	63805248	14832368	30305873	76922431
23580972	56034623	93604432	86094572	45197180	21758531	90846851	40780124	83524097	12967744	17301388
44327896	35421096	53191475	63812784	23414614	46797399	29352497	57133127	24934404	98848057	69968434
28920887	48355229	89630954	98753054	46569571	67023970	68741375	50691606	48471561	88982059	78041349
70684666	62494627	49119524	37188227	67815220	65981958	28617815	73545791	18358926	19016115	93694219
24827151	54009975	79670969	28750994	75917630	51539582	22109910	57585301	12843038	90847925	45450630
79596177	65079322	11044711	24092871	57361122	99870683	21094304	29586376	39274630	16630182	17064565
71674639	59765200	69342986	89833089	12960671	83129887	31467216	97718224	77151333	75863844	57467946
38075445	27263459	73923320	39804239	31937421	74661926	66933358	19226173	80740529	81946607	32015511
17343025	70067715	72135004	29359046	21755704	32560564	70013930	19859679	86606634	59663602	48234383
14284260	49724936	90178476	80389398	27575992	68428566	36491432	98197622	10792564	17810218	42583980
90360849	83837331	82459138	17324324	25542590	20797247	69729631	79114637	85740938	89842916	59317403
74056397	48112568	13885493	41753680	75075678	94290447	49316600	55866313	31435092	35816222	20241658
65461365	23827021	35356962	94765770	58506858	84834564	32973804	22357661	69818348	63424766	73526268
53976514	81327188	65251366	36995419	45718593	81892584	80708752	86696685	18085232	29092457	17806229
89933095	10412125	66002457	36410788	75561847	21875074	37734844	60727117	65880169	41339255	65590455
81486983	91671156	78219936	85772594	35282453	41421628	80182314	99493156	16097637	44617480	47584685
85878976	87315431	66359338	99979430	53948469	10161987	54398995	37506975	94149785	45714443	49107476
23967568	22770424	72549786	13494133	34492797	56545694	58534491	40024920	88410054	25885032	41630662
18362935	19128306	66514722	88803689	10629868	29484452	53914144	86851475	63354250	44827878	22849696
13179768	85406488	33463611	88388940	44702268	93962658	31006100	54734641	82738940	86061133	70685636
32761454	72188197	43288407	36808188	57229187	37415453	43204078	76227345	71468065	82323194	71590267
52647233	24000149	43171321	23675163	21726223	11151692	13105694	68291545	59516907	45320860	61928835
31120627	13345500	83388804	66403250	46365705	13169283	58254787	13872629	39243496	14747523	57059612
83243350	71403747	32520867	70584600	26456924	69451487	37246809	70657567	32226597	86726653	66110150
69089379	94248573	35003988	51006693	83909398	36916869	36651914	79818272	14478528	92183301	80080063
91811175	73142212	26382531	89227332	88682010	41331025	68676837	88147441	40768176	29609877	99839811
94805710	54621807	11598846	13501725	95596432	50782029	42851782	26945783	39938242	55039722	92851830
57443230	84811085	50487028	14311759	96474092	26347504	41417671	12056600	42509278	73153641	68540499
38468120	60365134	67260867	96401429	29509558	12959367	39617016	55361662	98590844	69436710	55244081
10042057	15262903	84636356	37872726	51328559	97333526	45948269	59211097	73638989	32337544	44130477
19475994	67925795	80608655	80686904	65197483	83879305	60255025	38078871	86821835	19125588	95788526
73047682	75489567	41938829	49015326	72390126	99354283	42344741	54304460	36362925	78639748	85461767
50795841	94145863	77067525	25270475	40077109	78669660	20192452	18510008	69260188	77317711	41813339
87969351	75540568	38466179	16976719	59084023	11373641	78139181	68989720	71201053	75519800	51446620
87978648	60011722	13294196	55039470	26341470	31959341	74104017	46665315	25868494	84760205	43661560
86029111	50585737	42658131	51912264	62748476	41698101	25677009	76471534	72048219	58082276	81736011
68270637	93716245	33969300	92561601	51746047	21414983	43247797	84607152	75032748	64729231	39994311
62667985	22764597	39193070	51905242	48232775	85233690	18755602	60564513	35883950	48487635	50957203
69823610	25229690	89476729	64452356	85850041	85441041	15038057	78233329	54178937	90903264	70052159
65810996	81821178	88081985	20953719	27862338	57199544	66751557	93175949	37956998	49382389	61538279
27669525	72462812	32020801	70039990	24182305	25331454	16208926	85327040	71778223	68028942	65785643
55659179	72723413	75470653	15981548	15905424	72871638	26545212	87002902	10330803	46177659	47301564
69183819	80438058	60508605	45409586	92676953	41590488	12340269	58490034	60953864	11848694	45662739
28726463	52236145	74249778	15654676	63719994	84619219	72767406	33487201	95476128	87655062	33078169
98258826	72411743	83392788	97887227	68160092	56712770	26937666	81906697	30197169	32646474	84951529
46561887	54568457	56723804	61114909	49021975	49672279	19380664	55640732	35599772	59666575	96303199
61197179	80225584	94326411	41034647	61912371	54081639	39325528	67082558	32843758	56226265	74539645
50705956	42141233	92323810	31933191	52727918	45484780	55679614	68749137	71369459	70678407	81532236
87899346	58603115	13704336	42011079	52626783	88006786	57898080	33097506	29829420	74815478	69560729
68807688	90151076	25427852	83948867	38746107	46932649	73736984	26237284	63781592	64440535	90965643
60335203	40939852	14305828	92145597	72394789	74940063	93232294	53400460	13885845	53750364	67793717
67144141	12078283	12411043	32099449	28232872	30116292	72990460	73547092	28549109	30272656	36152168
22104675	52124640	33084330	30050610	68935425	93422803	92549422	69855980	73790136	15668463	27829210
22862926	94802903	98849334	39908075	79700762	19184797	20962179	82653162	87067238	66495399	59263673
22798539	72528511	22458754	70767620	21143502	56119774	46940950	23248506	46776185	97258395	96110408
19079682	22003901	49130852	15219706	34609874	79563032	72979166	69160134	90068169	96710516	20867260
50763955	54492885	63697391	11295401	28257292	27634892	60619313	40356654	86428066	34916249	52890719
96377155	22588111	50670741	77099488	24899637	43365545	99366092	44460421			

TABLE 4. There are 1000 iterations in the iteration process; 112 integers with less than eight digits are eliminated, and the remaining 888 numbers are eight-digit random numbers R_i which are shown row-wise above for Algorithm 3.

The results show that the generated random numbers contain an approximately equal number of zeros and ones in their binary representation. These results are clearly illustrated in FIG. 5. Hence, the random numbers are uniformly distributed, indicating that the generated random numbers for 4, 6, and 8-digit cases are truly random.

4 digit random numbers

		VAR001
N		903
Uniform Parameters	Minimum	1000.00
	Maximum	9968.00
Most Extreme Differences	Absolute	.05
	Positive	.05
	Negative	.00
Kolmogorov-Smirnov Z		1.41
Asymp. Sig. (2-tailed)		.026

6 digit random numbers

		VAR001
N		907
Uniform Parameters	Minimum	101131.0
	Maximum	999511.0
Most Extreme Differences	Absolute	.02
	Positive	.02
	Negative	-.01
Kolmogorov-Smirnov Z		.64
Asymp. Sig. (2-tailed)		.810

8 digit random numbers

		VAR001
N		888
Uniform Parameters	Minimum	10042057
	Maximum	99979430
Most Extreme Differences	Absolute	.02
	Positive	.02
	Negative	-.02
Kolmogorov-Smirnov Z		.58
Asymp. Sig. (2-tailed)		.888

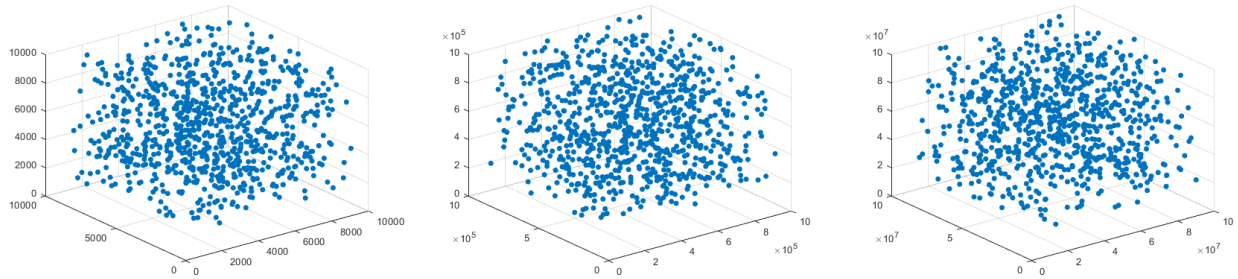
FIG. 5. The Kolmogorov-Smirnov test shows that the random numbers are uniformly distributed since the obtained p value .810 and .888 is greater than the significant p value of 0.5.

6.2. **Spectral Test for First 1000 Iterations.** The spectral test is a statistical test that evaluates the distribution of pseudo-random vectors in dimensions larger than the generator's genuine dimension. It is particularly useful for linear congruential generators (LCGs), which form lines or hyperplanes with all possible outputs. The test compares the distance between these planes, indicating the generator's quality. However, it is specific to LCGs and cannot be applied to other PRNG families, as it is designed for studying lattice structures. In general, the spectral test for the generated 1000 random numbers R_i, $1 \leq i \leq 1000$, involves plotting three consecutive random numbers $\{R_{i-2}, R_{i-1}, R_i\}$ for $1 \leq i \leq 1000$. However, not all the generated random numbers are guaranteed to have 4, 6, or 8 digits; sometimes they contain fewer digits. If a number has fewer than the required digits, it is automatically excluded from the list. Therefore, the spectral test for the generated 1000 random numbers R_i, $1 \leq i \leq 1000$, now becomes $\{R_i, R_j, R_k\}$, where $1 \leq i < j < k \leq 1000$. Among the generated numbers, 903 random numbers are 4-digit, 907 random numbers are 6-digit, and 888 random numbers are 8-digit numbers, as shown in FIG. 6.

7. Soundness and Order of Complexity

Existing PRNGs generate all random numbers from a single, unique seed, which leads to easy prediction of the random number; in contrast, the proposed PRNG generates all random numbers from a two-tier seed system: a user-selected seed at the first level and a server-selected seed associated with Pell curve solutions at the second level. This dual-seeding mechanism makes it significantly more difficult to predict the resulting random number.

With every iteration, the initial value i changes, meaning that the corresponding Lucas sequence values x_i and y_i also change. This implies that each generated intermediate value S_i and consequently the resulting random number R_i will also be distinct. The sum $x_i + y_i$ derived from the Pell curve solution is used to generate a unique integer for each seed. Since the Pell curve admits



(A) 903 random numbers of 4 digits

(B) 907 random numbers of 6 digits

(C) 888 random numbers of 8 digits

FIG. 6. Spectral Test shows that the 6 and 8-digit random numbers are uniformely distributed.

an infinite number of integer solutions, this ensures a diverse, vast pool of random numbers, each uniquely tied to a specific Pell curve solution.

Subsequently, the sum is hashed using the SHA-3 512 cryptographic algorithm. This hashing step ensures that the resulting binary sequence passes standard randomness checks such as the frequency test where the distribution of 1s and 0s is approximately balanced. The complexity of predicting the next random number is thus vastly increased, enhancing cryptographic strength. Hence, the random numbers produced can be considered as "almost truly random," offering a modern refinement to the existing perspectives in PRNG literature.

In terms of computational complexity: Generating Lucas sequences x_i and y_i via recursion has a time complexity of $O(2^n)$; Using iteration reduces this to $O(n)$; Modular multiplication has a time complexity of $O(\log^2 p)$; Computing S_i remains $O(n)$; Polynomial evaluations are $O(n^k)$; The nth root of unity computation is $O(n \log n)$; Calculating the hash $H(M)$ using SHA-3 512 is $O(\log^3 p)$; A non-trivial base conversion algorithm runs in $O(M(n) \log n)$. Given that $n < p$, the overall complexity of the algorithm is approximately $O(p)$, which demonstrates that the proposed approach is computationally efficient while ensuring strong cryptographic security.

8. Analysis of the Strength

A cryptographically secure pseudorandom number generator (CSPRNG) must pass rigorous statistical randomness tests, particularly the next-bit test. This test ensures that no polynomial-time algorithm can predict the $(k+1)$-th bit of the output sequence with success probability significantly greater than 50%. Passing the next-bit test implies that the generator also passes all other polynomial-time statistical tests for randomness.

CSPRNGs must also resist state compromise extension attacks, where even if part of the internal state is exposed, it should be computationally infeasible to reconstruct past or future outputs. High-quality randomness is achieved by drawing entropy from trusted sources such as the operating

system's randomness API. Additionally, CSPRNGs can "stretch" the available entropy across more output bits as needed, provided the entropy source is reliable and entropy extraction is properly implemented.

Existing PRNGs often exhibit vulnerabilities where the next output R_i for $i \geq 0$ can be predicted based on the previous output R_{i-1}. In contrast, our proposed method enhances security in multiple ways. The output random number R_i is 256 bits in size. Even if an attacker manages to capture this 256-bit output, reverse engineering the inputs is computationally impractical.

Specifically, R_i is derived from $(x_i + y_i) \bmod p$, and this sum is further processed using the SHA-3 512 hashing algorithm. Since SHA-3 512 produces a 512-bit output, and only 256 bits are exposed to the attacker, the remaining 256 bits remain unknown. Therefore, there are 2^{256} possibilities for the unknown bits, making brute-force attacks computationally infeasible with present-day computing capabilities.

Furthermore, even if an attacker could theoretically deduce $(x_i + y_i) \bmod p$, they would face the additional challenge of factoring this sum into its original components x_i and y_i, which are solutions of a Pell equation. Without knowledge of the index i or the parameters m and k, it is impossible to reconstruct the seed or infer future values. This ensures that the next random number R_{i+1} remains unpredictable, thereby maintaining the cryptographic strength and robustness of the proposed random number generator.

9. Security Analysis

9.1. **Direct Cryptanalytic Attack.** Direct cryptanalysis of the generator requires breaking SHA-3 512; however, currently no powerful cryptographic attacks against SHA-3 exist, making direct cryptanalytic attacks against the proposed PRNG infeasible. Therefore, direct cryptanalytic attacks are prevented.

9.2. **Input-Based Attacks.** An attacker controlling the inputs to a PRNG can force it to repeat the same output indefinitely, especially if they control some entropy samples. For example, if the first Lucas sequence is x_{i-1} and the next is x_i, the attacker could try to enforce:

$$ax_i = ax_{i-1} - I_{i-1} - 1 \pmod{p},$$

thus forcing the seed value and outputs to repeat. However, this attack fails if the user hashes their entropy samples before inputting them into the PRNG. Therefore, input-based attacks are effectively prevented.

9.3. **State Compromise Extension Attacks.** The PRNG remains strong against attacks provided there is sufficient entropy in the input samples. Therefore, state compromise extension attacks are prevented.

9.3.1. *Leaking Input Effects.* If the PRNG leaks effects of unguessable inputs into its outputs, it becomes vulnerable. Should an attacker compromise the PRNG's state, they could guess inputs without guessing the actual entropy sample, since future outputs depend only on previous outputs. This weakness would be mitigated if each new output R_{i-1} depended directly on ax_i and R_i. Even then, an attacker who knows the state could still guess the entropy sample, but losing state knowledge would reduce their advantage. Therefore, leaking input attacks are prevented.

9.3.2. *The Iterative Guessing Attack.* Once the state is exposed, the PRNG may be vulnerable to iterative guessing. For instance, an attacker knowing R_i and ax_i's entropy of only 20 bits could perform a 2^{20} search, generating a list of 2^{20} possible 160-bit outputs including I_i. The attacker only needs a function of the output to verify guesses, such as in a DSA signature where I_i is the secret parameter. Note that R_{i+1} can only be uniquely determined if I_i is known. Therefore, iterative Guessing attacks are prevented.

9.3.3. *Backtracking.* If an attacker knows R_i and I_{i-1}, they could theoretically recover R_{i-1}. However, prior knowledge of I_{i-1} is required, so the attacker gains no immediate benefit. While backtracking is possible in DSA PRNGs, it is not feasible in the proposed PRNG since S_i is hashed with SHA-3 512, making recovery of previous random numbers impossible. Therefore, backtracking attacks are prevented.

9.3.4. *Filling in the Gaps.* Suppose an attacker knows R_i, R_{i+2}, and I_{i+1} but needs I_i. They might attempt:

$$I_i = R_{i+1} + R_i - I_{i+1}.$$

However, finding S_{i+1} (and thus R_{i+1}) is computationally difficult, so filling in the gaps attack is not feasible.

10. Conclusion

The paper proposed a new pseudorandom number generator using the Lucas sequence over the Pell curve, which was then implemented in ATMs. The study presents one main algorithm and three sub-algorithms to generate high-security 4-digit, 6-digit, and 8-digit PIN numbers for ATM operations and e-commerce applications, ensuring that predicting these numbers is infeasible. The generator uses three inputs to enhance security and is based on the Lucas sequence. Experimental results were conducted for the first thousand iterations across three cases: four digits, six digits, and eight digits, depending on system requirements. The Kolmogorov–Smirnov test and spectral test results demonstrated that the generated random numbers are uniformly distributed. The proposed method is both efficient and secure.

Author Contributions: Conceptualization, E.R. and G.S.G.N.A.; methodology, E.R. and G.S.G.N.A.; writing-original draft preparation, E.R. and G.S.G.N.A.; writing reviews and editing, E.R. and G.S.G.N.A.; supervision, G.S.G.N.A.; All authors have read and agreed to the published version of the manuscript.

Acknowledgements: The authors would like to express their sincere gratitude to the editors and anonymous reviewers for their constructive comments and helpful suggestions, which have significantly enhanced the quality and clarity of this paper. The authors wish to thank the management of Vellore Institute of Technology (Vellore-632014) for their continuous support and encouragement to carry out this research work.

Conflicts of Interests: The authors declare that there are no conflicts of interest regarding the publication of this paper.

References

[1] D.E. Knuth, The Art of Computer Programming: Seminumerical Algorithms, Addison–wesley, 2014.
[2] Free Software Foundation, The GNU C Library (glibc) Manual, https://sourceware.org/glibc/manual.
[3] S.K. Park, K.W. Miller, Random Number Generators: Good Ones Are Hard to Find, Commun. ACM 31 (1988), 1192–1201. https://doi.org/10.1145/63039.63042.
[4] D. Crawford, Technical Correspondence, Commun. ACM 36 (1993), 105–110. https://doi.org/10.1145/159544.376068.
[5] P. L'Ecuyer, R. Touzin, Fast Combined Multiple Recursive Generators With Multipliers of the Form $a = \pm 2^q \pm 2^r$, in: Proceedings of the 2000 Winter Simulation Conference, pp. 683–689, 2000.
[6] M.E. O'Neill, PCG: A Family of Simple Fast Space-Efficient Statistically Good Algorithms for Random Number Generation, Technical Report, Harvey Mudd College, (2014). https://www.cs.hmc.edu/tr/hmc-cs-2014-0905.pdf.
[7] P. L'Ecuyer, Maximally Equidistributed Combined Tausworthe Generators, Math. Comput. 65 (1996), 203–213. https://doi.org/10.1090/s0025-5718-96-00696-5.
[8] P. L'Ecuyer, Tables of Maximally Equidistributed Combined LFSR Generators, Math. Comput. 68 (1999), 261–269. https://doi.org/10.1090/s0025-5718-99-01039-x.
[9] F. Panneton, P. L'Ecuyer, M. Matsumoto, Improved Long-Period Generators Based on Linear Recurrences Modulo 2, ACM Trans. Math. Softw. 32 (2006), 1–16. https://doi.org/10.1145/1132973.1132974.
[10] S. Vigna, A PRNG Shootout, https://prng.di.unimi.it.
[11] M. Matsumoto, T. Nishimura, Mersenne Twister, ACM Trans. Model. Comput. Simul. 8 (1998), 3–30. https://doi.org/10.1145/272991.272995.
[12] M. Saito, M. Matsumoto, Simd-Oriented Fast Mersenne Twister: A 128-Bit Pseudorandom Number Generator, Monte Carlo Quasi-Monte Carlo Methods 2006 (2008), 607–622. https://doi.org/10.1007/978-3-540-74496-2_36.
[13] M. Saito, M. Matsumoto, A Uniform Real Random Number Generator Obeying the IEEE 754 Format Using an Affine Transition, in: 8th International Conference on Monte Carlo and Quasi-Monte Carlo Methods in Scientific Computing (MCQMC '08), 2008.
[14] M. Saito, M. Matsumoto, A PRNG Specialized in Double Precision Floating Point Numbers Using an Affine Transition, Monte Carlo Quasi-Monte Carlo Methods 2008 (2009), 589–602. https://doi.org/10.1007/978-3-642-04107-5_38.
[15] S. Wolfram, Random Sequence Generation by Cellular Automata, Adv. Appl. Math. 7 (1986), 123–169. https://doi.org/10.1016/0196-8858(86)90028-x.
[16] P. Hortensius, R. McLeod, W. Pries, D. Miller, H. Card, Cellular Automata-Based Pseudorandom Number Generators for Built-In Self-Test, IEEE Trans. Comput. Des. Integr. Circuits Syst. 8 (1989), 842–859. https://doi.org/10.1109/43.31545.
[17] S. Das, B.K. Sikdar, A Scalable Test Structure for Multicore Chip, IEEE Trans. Comput. Des. Integr. Circuits Syst. 29 (2010), 127–137. https://doi.org/10.1109/tcad.2009.2034349.
[18] K. Bhattacharjee, D. Paul, S. Das, Pseudo-Random Number Generation Using a 3-State Cellular Automaton, Int. J. Mod. Phys. C 28 (2017), 1750078. https://doi.org/10.1142/s0129183117500784.

[19] M. Blum, S. Micali, How to Generate Cryptographically Strong Sequences of Pseudo Random Bits, in: Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali, Association for Computing Machinery, 2019: 227–240. https://doi.org/10.1145/3335741.3335751.

[20] C.J. Ballot, H.C. Williams, The Lucas Sequences: Theory and Applications, Springer, Cham, 2023. https://doi.org/10.1007/978-3-031-37238-4.

[21] M.R. Rizqi, A. Waluyo, G.M. Edgy, S.U. Sunaringtyas, Enhanced Authentication Mechanism for Automated Teller Machine (ATM) Through Implementation of Soft Two-Factor Authentication, in: Proceedings of the 3rd International Conference on Electronics, Communications and Control Engineering, ACM, New York, NY, USA, 2020, pp. 3-7. https://doi.org/10.1145/3396730.3396735.

[22] M.M. Fazili, M.F. Shah, S.F. Naz, A.P. Shah, Next Generation QCA Technology Based True Random Number Generator for Cryptographic Applications, Microelectron. J. 126 (2022), 105502. https://doi.org/10.1016/j.mejo.2022.105502.

[23] K. Bhattacharjee, S. Das, A Search for Good Pseudo-Random Number Generators: Survey and Empirical Studies, Comput. Sci. Rev. 45 (2022), 100471. https://doi.org/10.1016/j.cosrev.2022.100471.

[24] B. Ryabko, Asymptotically Most Powerful Tests for Random Number Generators, J. Stat. Plan. Inference 217 (2022), 1–7. https://doi.org/10.1016/j.jspi.2021.07.007.

[25] L. Yao, X. Wu, H. Zhang, DCDRO:A True Random Number Generator Based on Dynamically Configurable Dual-Output Ring Oscillator, Integration 93 (2023), 102053. https://doi.org/10.1016/j.vlsi.2023.102053.

[26] A.A. Pandit, A. Kumar, A. Mishra, Lwr-Based Quantum-Safe Pseudo-Random Number Generator, J. Inf. Secur. Appl. 73 (2023), 103431. https://doi.org/10.1016/j.jisa.2023.103431.

[27] J. Polo, H. López, C. Hernández, Random Number Generator Based on a Memristive Circuit, Heliyon 10 (2024), e26635. https://doi.org/10.1016/j.heliyon.2024.e26635.

[28] Q. Fan, F. Ran, L. Yan, Multi-Bit Per Cycle True Random Number Generator Based on Xor-Xnor Ring Oscillator Unit, Microelectron. J. 146 (2024), 106142. https://doi.org/10.1016/j.mejo.2024.106142.

[29] Y. Chen, H. Liang, L. Zhang, L. Yao, Y. Lu, High Throughput Dynamic Dual Entropy Source True Random Number Generator Based on FPGA, Microelectron. J. 145 (2024), 106113. https://doi.org/10.1016/j.mejo.2024.106113.

[30] E.B. Barker, J.M. Kelsey, Recommendation for Random Number Generation Using Deterministic Random Bit Generators, NIST Special Publication 800-90A, NIST, (2012). https://doi.org/10.6028/nist.sp.800-90a.