

Odd-Indexed Fibonacci Coprime Wheel Graph Labeling for Multi-Layer Modular Matrix Encryption with Dominator Coloring Key Partitioning and an Alphabet Fixed Graph Key

J Jeba Jesintha¹, J Sathya priya²

¹PG Dept. of Mathematics, Women's Christian College, University of Madras, Chennai.

²Dept. of Mathematics, Meenakshi College for Women, University of Madras, Chennai.

DOI: <https://doi.org/10.51244/IJRSI.2026.1308000111>

Received: 24 August 2026; Accepted: 29 August 2026; Published: 07 September 2026

ABSTRACT

This paper proposes a multi-layer symmetric encryption scheme built on an odd-indexed Fibonacci coprime graceful labeling of a wheel graph. Each letter of a fixed, pre-agreed alphabet is assigned a distinct Fibonacci number at an odd index determined by a shared secret seed, and the wheel's hub is assigned a Fibonacci number at a prime index exceeding all rim indices. Using the identity $\gcd(F_m, F_n) = F_{\gcd(m, n)}$, we prove that every rim edge and every spoke edge of the resulting graph is automatically coprime. A genuine Minimum Spanning Tree is extracted via Kruskal's algorithm from this fixed graph and combined with a message-specific, order-encoding diagonal to build a structural key matrix. Building on this foundation, two further contributions are made. First, we prove that the dominator chromatic number of a wheel graph equals its ordinary chromatic number for every n , because the hub's universal adjacency forces it into a singleton color class; this guarantees a specific number of independent, seed-derived key blocks for any message, hardening the cipher against known-plaintext attack without any extra shared secret. Second, and most significantly, we remove the plaintext-dependence of the underlying graph itself: rather than building a new wheel graph for every message (which would make the graph function as an undesirable per-message session key), we fix a single wheel graph once, over a pre-agreed alphabet, so that the graph, its minimum spanning tree, and every key block are entirely message-independent and reusable across arbitrarily many messages. Only the diagonal of the structural matrix, encoding which letters occur and in what order, varies from message to message. The message-specific diagonal itself uses a positional packing scheme so that every occurrence of every letter, including repeated letters, is individually recoverable, rather than only the set of distinct letters used and their first-appearance order. The complete encryption and decryption procedure is illustrated on two worked examples, MATHEMATICS and CRYPTOGRAPHY, sharing the same fixed graph, keys, and permutation, with full matrices and an explicit computational verification that decryption exactly recovers each full plaintext message, letter for letter. Key generation is performed via HMAC-SHA256-based derivation rather than a general-purpose pseudorandom generator, the modulus is fixed once from worst-case bounds rather than derived per message (eliminating a message-length side channel), and a quantified, conservative known-plaintext lower bound is given for the scheme's key material. We conclude with a discussion of these security properties and the scheme's remaining, honestly stated limitations.

Keywords- Odd Fibonacci numbers, coprime graceful labeling, wheel graph, dominator coloring, minimum spanning tree, modular matrix cryptograph, graph-based key generation.

INTRODUCTION

Graph labeling has a long history of application to cryptography, since a labeled graph can simultaneously encode a message's structure and a numerical key derived from the labeling rule itself. Rosa's original graceful labelings, Rosa (1967)[5] and their many variants, including prime and prime-graceful labelings, Selvarajan & Subramoniam(2018)[6], Panma & Rochanakul(2021)[4], assign integers to vertices so that induced edge weights satisfy an arithmetic condition.

Nayana and Sobha (2023)[3] proposed an encoding/decoding scheme building a weighted graph from a plaintext message using a prime-graceful labeling, but their illustrative example adds edges between letters that are not adjacent in the plaintext string without stating this design choice, and their vertex labels come directly from alphabet position, carrying no secret information.

This paper addresses this gap by building the graph as a wheel graph with a genuine Minimum Spanning Tree step and Fibonacci-seeded vertex labels, hardening the key structure with a dominator-coloring key partition, and most significantly, fixing the graph itself once, over a pre-agreed alphabet and a shared seed, so that it is reused unchanged for every message rather than rebuilt from each plaintext's specific letters. Rebuilding the graph per message would make the graph itself function as an undesirable per-message session key that must be communicated or reconstructed alongside the ciphertext, fixing it once and for all avoids this entirely. Only the diagonal of the structural matrix varies per message, isolating all plaintext-dependent information into the single part of the construction designed to carry it.

Preliminaries and Definitions

Definition 2.1 The Fibonacci sequence is a series of numbers starting with 0 and 1, where each succeeding number is the sum of the two preceding numbers. Hence the Fibonacci sequence is 0, 1, 1, 2, 3, 5, 8, 13, 21 and so on. It can be obtained by the recursive formula $F_n = F_{n-1} + F_{n-2}$, for $n \geq 2$, with initial values $F_0 = 0$ and $F_1 = 1$.

Lemma 2.2 (Fibonacci gcd identity)

For all positive integers m and n , $\gcd(F_m, F_n) = F_{\gcd(m, n)}$

Hence

- consecutive odd-indexed Fibonacci numbers are always coprime
- A Fibonacci number at a prime index exceeding every other index used is coprime to all of them.

Definition 2.3 A wheel graph W_n consists of a cycle C_n (the rim) plus one hub vertex adjacent to every rim vertex (the spokes). A wheel graph has $n+1$ vertices and $2n$ edges.

Definition 2.4 Odd-indexed Fibonacci coprime labeling of a wheel graph W_n given a seed $s \geq 0$ and n rim vertices is done by assigning vertex v_i the Fibonacci number F_{s+2i-1}

Definition 2.5 A minimum spanning tree is a spanning subgraph connecting all vertices with $|V|-1$ edges of minimum total weight and no cycle, calculated via Kruskal's algorithm.

Definition 2.6 A dominator coloring of a graph G is a proper vertex coloring in which every vertex either is alone in its color class or is adjacent to every vertex of at least one color class. The dominator chromatic number $\chi_d(G)$ is the minimum number of classes in such a coloring [Bresar, Klavzar, Rall].[1,2]

Definition 2.7 In classical cryptography, Plaintext is the original message that needs to be sent and Ciphertext is its encrypted form, a Key is the shared secret required to transform between the two.

PROPOSED METHODOLOGY

The scheme is built in two stages. Stage A (Section 3.1-3.2) is performed once, when the sender and receiver first agree to communicate, and produces a graph and key structure that is entirely independent of any specific message. Stage B (Section 3.3 onward) is performed once per message and produces the ciphertext.

Stage A, Step 1 -- Fix a shared alphabet and seed

Sender and receiver agree in advance on a finite ordered alphabet of n letters covering every letter either party might need to send (for illustration in this paper, $n = 13$, the union of the distinct letters in our two worked examples: A, C, E, G, H, I, M, O, P, R, S, T, Y, in a production deployment this would instead be the full 26-letter English alphabet, or any fixed symbol set required by the application); and a shared secret seed s .

On the alphabet ordering

The rim order of the fixed alphabet is simple lexicographic (alphabetical) order in this paper, chosen purely for readability and reproducibility - it carries no cryptographic significance of its own, since the labeling, wheel graph, and spanning tree are deterministic functions of whichever fixed order is agreed upon. This order is communicated once, alongside the seed, as part of the same long-term shared setup; it need not be secret, since security rests on the seed-derived key blocks (Stage A, Step 5) and LT_2 , not on the alphabet order itself. A deployment wanting to avoid publishing even this order could instead derive it from a keyed permutation of the alphabet, seeded the same way as the key blocks, at the cost of an extra shared parameter; this is a possible hardening but is not required for the security properties claimed in this paper.

Stage A, Step 2 -- Odd-indexed Fibonacci labeling of the fixed alphabet

Assign the i -th letter of the fixed alphabet the value F_{s+2i-1} assign the hub F_p , where p is the smallest prime exceeding the largest rim index. By Lemma 2.2, every rim edge and every spoke is coprime by construction, for any n .

Stage A, Step 3 -- Build the fixed wheel adjacency matrix M_1

Connect rim neighbours and hub spokes; weight = |difference of endpoint values|. Because this graph depends only on the fixed alphabet and the seed, M_1 is identical for every message ever sent under this agreement - it is computed once and never varies.

Stage A, Step 4 -- Fixed Minimum Spanning Tree

Kruskal's algorithm on M_1 's $2n$ edges produces one fixed n -edge spanning tree. Its off-diagonal weights form the off-diagonal of the structural matrix M_2 , and, like M_1 , never change between messages.

Theorem 1 (Dominator-coloring key partition)

For every $n \geq 3$, $\chi_d(W_n) = \chi(W_n)$, where $\chi(W_n) = 3$ if n is even and 4 if n is odd.

Proof

The hub of W_n is adjacent to every rim vertex. Hence in any proper coloring, the hub cannot share a color class with any other vertex, the hub is therefore always alone in its own class. This gives two facts for free, every rim vertex is adjacent to the hub, so every rim vertex dominates the hub's singleton class trivially and the hub is adjacent to every rim vertex, so the hub dominates every rim color class in its entirety, regardless of how the rim is partitioned. Thus any proper coloring of W_n already satisfies the dominator condition for every vertex, giving $\chi_d(W_n) = \chi(W_n)$. The classical result $\chi(W_n) = 3$ (n even) or 4 (n odd) follows since the rim cycle C_n needs 2 colors if n is even and 3 if n is odd, and the hub needs one further color distinct from all of them.

Remark 1 (Minimality of the coloring used in Stage A)

Theorem 1 is stated as an equality, $\chi_d(W_n) = \chi(W_n)$, not merely as an upper bound. This is important because the number of independent key blocks generated in Stage A equals the number of color classes used, and Stage A always uses the standard MINIMUM proper coloring of W_n - the rim cycle 2-colored by alternation when n is even (3-colored when n is odd), plus one further color for the hub. Since Theorem 1 proves this minimum

proper coloring already satisfies the dominator condition, it is simultaneously a minimum dominator coloring: no proper dominator coloring of W_n can use fewer classes. Stage A therefore always generates the smallest possible number of key blocks under this construction (3 for even n , 4 for odd n), rather than an arbitrary valid-but-suboptimal dominator coloring with unnecessarily many classes.

Stage A, Step 5 -- Fixed per-class key generation

Let the $\chi_d(W_n)$ dominator-coloring classes be $V_1, V_2, \dots, V_k$. Generate, once, an independent invertible upper-triangular key matrix K_i for each class, seeded deterministically by (s, i) , so both parties can regenerate every K_i from the shared seed alone. Form $K_1' = P_1^T \cdot \text{diag}(K_1, \dots, K_k)$. P_i , where P_i is the (public, structure-derived) permutation grouping vertices by class. K_1' , like M_1 and the spanning tree, is fixed once and reused for every message.

Key derivation: an HMAC-based cryptographic key derivation function

Each K_i , and likewise LT_2 (Stage A, Step 6), is generated using an HMAC-SHA256 counter-mode expansion in the style of HKDF-Expand (RFC 5869), rather than a general-purpose pseudorandom generator. General-purpose generators such as Mersenne Twister are not cryptographically secure: their internal state can, in principle, be reconstructed from a sufficient run of output, which is an inappropriate property for key-generating material. For class i , the byte stream $\text{HMAC-SHA256}(\text{key} = s, \text{message} = "K_i | \text{counter}")$ is computed for counter = 1, 2, 3, ... and each block is reduced modulo q to fill K_i 's entries in row-major order, forcing any zero landing on the diagonal to 1 to preserve invertibility. LT_2 is generated by the same construction under the label " LT_2 ". Both K_i and LT_2 remain fully deterministic from the shared seed s alone - nothing new needs to be transmitted or agreed - and, being HMAC-derived, are computationally indistinguishable from random, unlike output from a general-purpose PRNG.

Stage A, Step 6 -- Fixed permutation and second key

Fix P , a cyclic shift of all $n+1$ vertex positions by s places generated once from the shared seed and reused for every message and LT_2 , (Step-5 above), is likewise fixed once and reused unchanged.

Stage B -- Per-message encryption

Each letter of the fixed alphabet is assigned a single diagonal value in M_2 that must, on its own, encode every position at which that letter occurs in the message - not merely whether it occurs, and not merely the order of its first appearance, since a repeated letter (e.g. the two occurrences of 'M' in MATHEMATICS) would otherwise collapse to a single entry and be lost. This is achieved with a positional packing scheme, for a letter l , let its 1-indexed occurrence positions in the message be $p_1, p_2, \dots, p_r$ (up to a fixed capacity R of occurrences per letter, $R = 3$ in our examples), and encode them as a single integer value $(l) = p_1 \times B^{R-1} + p_2 \times B^{R-2} + \dots + p_r \times B^0$, using a fixed public base B exceeding the longest message length anticipated ($B = 21$ in our examples, supporting messages up to 20 characters with up to 3 repeats of any letter). A letter that does not occur receives value $(l) = 0$. Build M_2 by taking the fixed spanning-tree off-diagonal from Stage A and placing these packed positional values (and the seed s in the hub's position) on the diagonal. This positional encoding, rather than order-of-first-appearance alone, is what allows the full plaintext - including every repeated letter - to be reconstructed exactly at decryption. Packing and unpacking use only B and R , both public and fixed once for the whole scheme, this is the only step in the entire construction that depends on the plaintext.

Choice of the modulus q

The modulus q is fixed once, during Stage A, entirely independently of any message, so that its magnitude cannot leak information about message length to an eavesdropper. Specifically, $q_{\max}$ is chosen as the smallest prime exceeding four times a worst-case bound on any M_3 entry the fixed alphabet could ever produce (computed from the fixed M_1 's largest entry and the largest value the positional-packing scheme in Section 3.2 could ever place on a diagonal or the fixed spanning tree's largest edge weight), advanced if necessary so that $q_{\max}$ does not divide $\det(M_1)$. The same $q_{\max}$ is then used, unchanged, for every message. In the theoretically possible but extremely

rare case that $q_{\max}$ divides a specific message's $\det(M_2)$, the sender and receiver fall back to a single pre-agreed alternate prime $q_{\max_alt}$, a shared one-bit flag, not a message-derived numeric value, so no length information is leaked even in this edge case. In our worked examples, $q_{\max} = 62,728,147,919,399$, identical for both MATHEMATICS and CRYPTOGRAPHY despite their different lengths, confirming computationally that no length side-channel is present.

Encryption algorithm

- 1. Look up the fixed M_1 , spanning-tree off-diagonal, K_1' , P , and LT_2 and $q_{\max}$ from Stage A (no recomputation needed).
- 2. Build this message's diagonal (packed occurrence-position value per letter as described above, 0 if unused, seed in the hub position) and complete M_2 .
- 3. Compute $M_3 = M_1 \times M_2$
- 4. Use $q = q_{\max}$ (fixed; see previous section) unless the rare fallback condition applies.
- 5. Compute $C_1 = (M_3 \times K_1') \bmod q$
- 6. Compute $C_2 = (P \times C_1 \times P^T) \bmod q$.
- 7. Compute $C_3 = (C_2 \times LT_2) \bmod q$.
- 8. Compute $C_{\text{Final}} = (C_3 + M_1) \bmod q$ and transmit C_{Final} together with q . Neither M_1 , K_1' , P , nor LT_2 need to be transmitted, since both parties already hold them from Stage A.

Decryption algorithm

- 1. $C_3 = (C_{\text{Final}} - M_1) \bmod q$.
- 2. $C_2 = (C_3 \times LT_2^{-1}) \bmod q$.
- 3. $C_1 = (P^{-1} \times C_2 \times (P^{-1})^T) \bmod q$, using $P^{-1} = P^T$.
- 4. $M_3 = (C_1 \times (K_1')^{-1}) \bmod q$
- 5. $M_2 = (M_1^{-1} \times M_3) \bmod q$.
- 6. Read the diagonal of M_2 , for each fixed-alphabet letter, unpack its value in base B into up to R occurrence positions (discarding zero-padding), a zero value means the letter did not occur at all. The hub's entry returns the seed s as an integrity check.
- 7. Place each recovered letter at every one of its recovered positions to reconstruct the plaintext in full, including all repeated letters.

Key material to be shared

- Long-term (Stage A, established once): the fixed alphabet and its order; the seed s ; the resulting M_1 , spanning tree, K_1' , P , LT_2 , and $q_{\max}$, all derivable by both parties from the alphabet and the seed alone.
- Per-message (Stage B): only the ciphertext C_{Final} needs to be sent; $q_{\max}$ is already shared and no longer varies per message.

Computational Complexity

Stage A is performed once and its cost is amortised over every message subsequently sent. Building M_1 costs $O(n^2)$ for n rim vertices; Kruskal's algorithm on the $2n$ edges of W_n costs $O(n \log n)$, generating the dominator-coloring key blocks via HMAC-SHA256 expansion costs $O(N^2)$ hash evaluations in total across all classes, where $N = n+1$. None of this is repeated for subsequent messages.

Stage B, performed once per message of length L , costs $O(L)$ to build the positional-packed diagonal, $O(N^3)$ for the dense matrix product $M_3 = M_1 \times M_2$ and each subsequent matrix multiplication in the pipeline, and $O(N^2)$ for the final additive mask. Decryption costs the same $O(N^3)$ per matrix inversion step, except inverting K_1' where the block-diagonal structure is exploited directly, inverting each of the k class blocks separately costs $\sum(\text{size } i^3)$ scalar multiplications rather than N^3 for a dense inverse of the same overall dimension. For our 13-letter, 4-class example ($N = 14$, class sizes 6, 6, 1, 1), this is 434 scalar multiplications instead of 2,744 - roughly a 6.3x reduction for this one step, growing more favourable as class sizes shrink relative to N .

On the concern that scaling from a 13-letter to a full 26-letter alphabet produces very large Fibonacci numbers (for instance F_{53} for the hub with seed $s = 4$): this is true in absolute value - $F_{53} = 53,316,291,173$ and the 26-letter hub value $F_{59} = 956,722,026,041$ - but the relevant cost driver for arbitrary-precision arithmetic is bit length, not decimal magnitude, and Fibonacci bit length grows only LINEARLY in the index (approximately 0.694 bits per index, since F_n is proportional to the golden ratio raised to n). Concretely, F_{53} requires 36 bits, F_{59} requires 40 bits, and even a 100-letter alphabet (hub index 211) requires only F_{211} , a 146-bit, 44-digit number -- all comfortably within the native arbitrary-precision integer types used throughout this paper's reference implementation (Python's sympy), with no special large-integer handling required. The scheme therefore scales to realistic alphabet sizes without a computational barrier; it would only become a genuine practical concern for alphabets of many thousands of symbols, which is outside any anticipated use case for letter-based messages.

Illustrative Examples

Both examples below share the identical fixed alphabet $\{A, C, E, G, H, I, M, O, P, R, S, T, Y\}$, seed $s = 4$, fixed graph M_1 , fixed spanning tree, fixed dominator-coloring classes, and fixed keys K_1' , P , LT_2 , none of this is recomputed between examples.

Shared dominator-coloring classes (fixed, computed once)

Color class	Vertices	Independent key size
Class 1	A, E, H, M, P, S	6
Class 2	C, G, I, O, R, T	6
Class 3	Y	1
Class 4	HUB	1

EXAMPLE 1: "MATHEMATICS"

Message-specific unique letters in order of first appearance: M, A, T, H, E, I, C, S.

Message-specific diagonal values over the fixed alphabet (packed occurrence positions, base 21)

Letter	A	C	E	G	H	I	M	O	P	R	S	T	Y
Packed value	1029	4410	2205	0	1764	3969	567	0	0	0	4851	1491	0

M₂ (fixed spanning-tree off-diagonal + this message's diagonal)

	A	C	E	G	H	I	M	O	P	R	S	T	Y	HUB
A	1029	8	0	0	0	0	0	0	0	0	0	0	0	0
C	8	4410	21	0	0	0	0	0	0	0	0	0	0	0
E	0	21	2205	55	0	0	0	0	0	0	0	0	0	0
G	0	0	55	0	144	0	0	0	0	0	0	0	0	0
H	0	0	0	144	1764	377	0	0	0	0	0	0	0	0
I	0	0	0	0	377	3969	987	0	0	0	0	0	0	0
M	0	0	0	0	0	987	567	2584	0	0	0	0	0	0
O	0	0	0	0	0	0	2584	0	6765	0	0	0	0	0
P	0	0	0	0	0	0	0	6765	0	17711	0	0	0	0
R	0	0	0	0	0	0	0	0	17711	0	46368	0	0	0
S	0	0	0	0	0	0	0	0	0	46368	4851	121393	0	0
T	0	0	0	0	0	0	0	0	0	0	121393	1491	317811	0
Y	0	0	0	0	0	0	0	0	0	0	0	317811	0	832040
HUB	0	0	0	0	0	0	0	0	0	0	0	0	832040	4

M₃ = M₁ x M₂

	A	C	E	G	H	I	M	O	P	R	S	T	Y	HUB
A	64	35280	168	0	0	0	0	0	0	0	0	163426043664	1120145498560	427860322016

C	823 2	505	463 05	115 5	0	0	0	0	0	0	0	0	11201 38842 240	5385 024
E	168	926 10	346 6	0	792 0	0	0	0	0	0	0	0	11201 21369 400	5384 940
G	0	115 5	121 275	237 61	254 016	542 88	0	0	0	0	0	0	11200 75607 200	5384 720
H	0	0	792 0	0	162 865	149 631 3	372 099	0	0	0	0	0	11199 55793 440	5384 144
I	0	0	0	542 88	665 028	111 629 8	559 629	2550 408	0	0	0	0	11196 42114 360	5382 636
M	0	0	0	0	372 099	391 740 3	765 122 5	0	1748 0760	0	0	0	11188 20890 880	5378 688
O	0	0	0	0	0	255 040 8	146 512 8	5244 2281	0	1198 1491 5	0	0	11166 70899 520	5368 352
P	0	0	0	0	0	0	174 807 60	0	3594 4474 6	0	8212 2364 8	0	11110 42148 920	5341 292
R	0	0	0	0	0	0	0	1198 1491 5	0	2463 6709 45	2249 3116 8	5628 7506 24	10963 05888 480	5270 448
S	0	0	0	0	0	0	0	0	8212 2364 8	0	1688 6251 873	1809 9696 3	10963 05888 483	5084 976
T	0	0	0	0	0	0	0	0	0	5628 7506 24	5888 7744 3	1157 4009 2170	95672 20260 40	2644 3606 3844
Y	529 136 496	411 379 2	0	0	0	0	0	0	0	0	3858 0030 723	4738 5620 1	79329 43933 21	3328 160
H U B	139 607 570 4	597 603 000 7	307 075 945 1	267 872 109	307 557 086 7	717 556 740 7	555 854 984 9	1250 8092 543	3241 5451 452	8259 4947 445	2068 4570 0303	4204 6601 5173	36543 52961 61	6922 9056 1600

Modulus and ciphertext

$q = 62728147919399$.

C_{Final} (transmitted):

	A	C	E	G	H	I	M	O	P	R	S	T	Y	HU B
A	5544 6862 9885 7	1547 9864 1118 23	5157 9490 1804 56	1576 8146 4894 20	1792 0489 4086 44	4471 1318 8745 2	5669 1661 5153 47	5521 1359 3572 02	4436 7120 7732 01	9615 9716 4858 0	3468 1213 9721 26	8024 8247 9860 3	4724 1471 6683 46	1346 264
C	6109 2511 9633 51	1629 7044 2931 14	3173 4009 0212 05	3038 1537 8117 07	8556 8944 8665	1705 9491 5059 7	8835 6411 0249 7	5979 7768 7069 83	2480 3302 1886 7	2515 4274 3865 93	5481 4541 9715 53	2621 8032 1332 80	1817 4128 9220 33	5340 7763 8753 44
E	4828 3163 9595 83	2136 5930 0384 60	5692 8126 4882 87	2222 1358 5244 6	1111 5388 8514 73	4591 9131 2912 96	5742 2158 8334 05	2017 0198 6089 5	5019 7668 4364 78	9904 5223 5906	0	0	0	1346 235
G	4467 4158 9024 73	1823 3902 2803 95	4277 1592 7206 47	6251 1676 7897 40	5566 5861 0056 86	7593 4885 6131 4	1930 2038 8649 59	3765 7960 1659 39	2573 5288 7658 22	3310 4984 4791 62	0	0	0	1346 180
H	2202 2387 5823 86	1384 5492 0714 95	1265 6361 1912 38	6599 2820 5037 2	3925 4894 9610 28	3356 8869 6107 21	5428 5780 5927 08	3180 0049 0097 24	4500 5801 6092 39	4027 6825 8169 99	0	0	0	1346 036
I	2667 3180 1728 68	3040 4919 7031 81	2911 0614 2364 38	1820 5238 1587 11	7614 0902 1170 8	3313 3597 1324 83	6129 1635 5174 47	4321 5720 2758 1	1823 4239 7134 8	2967 7817 5872 54	0	0	0	1345 659
M	2983 5493 9747 8	3621 8528 7149 08	5742 9580 4006 79	3384 8800 5006 79	4537 4068 5738 35	1827 0845 0927 79	9348 9966 7974 3	5872 0377 4213 09	5053 4023 0009 5	5343 7099 4795 81	0	0	0	1344 672
O	2956 3637 8747 44	2159 1127 8263 81	4359 3433 7515 96	6057 1680 8216 57	5200 4998 2531 22	4860 7981 3640 63	1092 5414 4874 28	4688 6818 9592 87	4976 8154 5134 64	4703 4589 0424 61	0	0	0	1342 088
P	7748 4651 0280 3	5679 5222 9068 80	1118 4957 7522 06	3398 4881 9880 47	3549 8236 4171 47	2006 9087 6747 91	1326 3198 2728 90	1076 8459 6611 35	3627 8929 3521 34	3808 0435 7100 44	4805 8061 8603 73	4173 9447 4732 19	7561 8178 7475 0	3769 3150 7967 82

R	1030 5313 8788 64	6589 8225 2999 0	1259 4885 4293 02	6889 0077 4022 4	1558 7993 3892 58	5163 0807 5851 66	4473 2842 6559 8	4612 7708 6828 12	2514 5879 2177 39	1629 4576 8416	3636 8293 4369 44	5269 4691 0007 82	1592 7864 2319 91	2481 6042 1733 72
S	1918 4696 0659 67	5297 0752 0468 26	4772 0844 3805 14	3205 0245 1914 90	5501 1914 0443 57	2449 4595 7905 68	2728 1311 2063 50	3898 9512 8480 44	6435 4105 2451 7	1275 4666 3325 35	6122 6634 7792 04	6161 7710 9170 48	4976 1983 7435 23	5326 4954 9292 77
T	2109 8545 5814 24	5889 4544 8379 14	4214 9657 6523 71	4957 0523 3321 39	1293 6188 0327 47	7874 4176 4489	4477 3385 5942 58	4738 6544 1052 25	1090 1614 9132 87	3923 2030 7962 84	4111 2116 3714 09	1208 4860 2429 80	4311 3659 0417 34	3142 4149 8505 19
Y	5200 3383 6631 21	4271 2225 8429 43	6169 8107 8136 48	5063 8861 7689 35	6244 4999 7623 37	5191 8182 6538 34	3595 8519 7712 11	4300 3809 2733 39	5877 9266 8472 72	7232 5258 4117 3	3195 6656 2033 47	3717 7171 8754 28	3889 4565 9224 99	5356 9300 7952 03
H U B	6101 2295 8393 24	4535 7756 9438 07	3456 6288 9936 79	7701 6647 2312 6	3201 2271 7305 56	5293 5858 9731 48	1852 0411 8922 17	4134 6150 9979 04	1630 3418 0233 6	2874 9018 8436 16	3894 1250 1692 95	2606 0570 5133 19	4044 5434 8563 44	2033 1425 0450 96

Decryption verification

- Decryption exactly recovered every diagonal entry (packed values and the seed)
- Full plaintext reconstructed by placing each recovered letter at every recovered position: "MATHEMATICS"

EXAMPLE 2: "CRYPTOGRAPHY"

Message-specific unique letters in order of first appearance: C, R, Y, P, T, O, G, A, H.

Message-specific diagonal values over the fixed alphabet (packed occurrence positions, base 21)

Lett er	A	C	E	G	H	I	M	O	P	R	S	T	Y
Pack ed valu e	3969	441	0	3087	4851	0	0	2646	1974	1050	0	2205	1575

M₂ (fixed spanning-tree off-diagonal + this message's diagonal)

	A	C	E	G	H	I	M	O	P	R	S	T	Y	HUB
A	396 9	8	0	0	0	0	0	0	0	0	0	0	0	0

C	8	44 1	2 1	0	0	0	0	0	0	0	0	0	0	0
E	0	21	0	55	0	0	0	0	0	0	0	0	0	0
G	0	0	5 5	308 7	144	0	0	0	0	0	0	0	0	0
H	0	0	0	144	485 1	37 7	0	0	0	0	0	0	0	0
I	0	0	0	0	377	0	987	0	0	0	0	0	0	0
M	0	0	0	0	0	98 7	0	258 4	0	0	0	0	0	0
O	0	0	0	0	0	0	258 4	264 6	6765	0	0	0	0	0
P	0	0	0	0	0	0	0	676 5	1974	1771 1	0	0	0	0
R	0	0	0	0	0	0	0	0	1771 1	1050	46368	0	0	0
S	0	0	0	0	0	0	0	0	0	4636 8	0	12139 3	0	0
T	0	0	0	0	0	0	0	0	0	0	12139 3	2205	31781 1	0
Y	0	0	0	0	0	0	0	0	0	0	0	31781 1	1575	83204 0
HU B	0	0	0	0	0	0	0	0	0	0	0	0	83204 0	4

$$M_3 = M_1 \times M_2$$

	A	C	E	G	H	I	M	O	P	R	S	T	Y	HUB
A	64	352 8	168	0	0	0	0	0	0	0	0	1634 2604 3664	11209 55401 360	4278 6032 2016
C	3175 2	505	0	1155	0	0	0	0	0	0	0	0	11201 38842 240	5385 024
E	168	926 1	346 6	1697 85	7920	0	0	0	0	0	0	0	11201 21369 400	5384 940

G	0	115 5	0	2376 1	6985 44	5428 8	0	0	0	0	0	0	11200 75607 200	5384 720
H	0	0	792 0	4445 28	1628 65	0	3720 99	0	0	0	0	0	11199 55793 440	5384 144
I	0	0	0	5428 8	1828 827	1116 298	0	2550 408	0	0	0	0	11196 42114 360	5382 636
M	0	0	0	0	3720 99	0	7651 225	6837 264	1748 0760	0	0	0	11188 20890 880	5378 688
O	0	0	0	0	0	2550 408	0	5244 2281	1335 4110	1198 1491 5	0	0	11166 70899 520	5368 352
P	0	0	0	0	0	0	1748 0760	1790 0190	3594 4474 6	1859 6550	8212 2364 8	0	11110 42148 920	5341 292
R	0	0	0	0	0	0	0	1198 1491 5	3496 1514	2463 6709 45	0	5628 7506 24	10963 05888 480	5270 448
S	0	0	0	0	0	0	0	0	8212 2364 8	4868 6400	1688 6251 873	2676 7156 5	10963 05888 483	5084 976
T	0	0	0	0	0	0	0	0	0	5628 7506 24	0	1157 4009 2170	95722 25783 65	2644 3606 3844
Y	2040 9550 56	411 379 2	0	0	0	0	0	0	0	0	3858 0030 723	7007 7325 5	79329 43933 21	3328 160
H U B	5354 0918 64	632 739 943	102 311 276	4423 5297 69	7230 7839 99	1834 6468 36	4796 1208 25	1605 9257 391	3505 1379 054	8397 8440 045	2006 7889 5659	4212 8700 8787	36674 57591 61	6922 9056 1600

Modulus and ciphertext

$q = 62728147919399$.

C_{Final} (transmitted):

	A	C	E	G	H	I	M	O	P	R	S	T	Y	HU B
--	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	-----------------

A	2939 4812 7229 16	1643 2299 7354 56	2855 6394 7702 10	2265 5456 4094 31	6718 0682 0507 7	4075 3740 4535 50	8341 1690 7424 7	3739 0630 5527 31	2486 9131 8217 42	3798 6879 0063 29	1414 1123 2176 42	6047 8726 5864 58	1882 8174 5184 0	5928 7563 2943 28
C	7528 6092 8342 4	4533 8831 8193 49	5450 4117 6420 55	1977 6770 2319 13	4221 2345 9650 54	8626 2249 7345 7	7655 3355 4992 7	5979 7768 7069 83	2480 3302 1886 7	2515 4274 3865 93	5481 4541 9715 53	2621 8032 1332 80	1817 4128 9220 33	5340 7763 8753 44
E	1838 4083 8947 26	3104 1736 2206 09	1442 8993 8223 1	3854 5337 8732 75	5225 8643 5739 44	1892 4373 3993	4811 6616 9782 8	1456 2357 5258 80	5019 7668 4364 78	9904 5223 5906	0	0	0	1346 235
G	3095 2283 2432 56	9929 2818 5132 1	2627 6677 6208 54	2262 3537 2627 32	2716 8404 6935 86	5623 2167 1735 89	1514 1012 3832 7	3765 7960 1659 39	2573 5288 7658 22	3310 4984 4791 62	0	0	0	1346 180
H	1291 2719 0929 3	2941 0048 1563 51	4031 2589 6660 37	4053 9685 2542 22	3596 9755 3800 59	4302 6282 3656 79	2653 2401 5473 67	1312 6038 2872 16	4500 5801 6092 39	4027 6825 8169 99	0	0	0	1346 036
I	7875 8980 2889 3	2155 6550 4919 21	5272 0312 3780 72	1038 5411 4663 86	3247 8656 2661 67	9725 6939 3863 8	1830 0067 4409 32	4321 5720 2758 1	1823 4239 7134 8	2967 7817 5872 54	0	0	0	1345 659
M	1616 8124 6577 40	2051 9349 8989 2	3493 2638 6169 90	2971 1209 4098 73	2774 7420 1688 67	4996 7902 5738 57	5030 9344 1282 94	5097 6226 7849 71	5053 4023 0009 5	5343 7099 4795 81	0	0	0	1344 672
O	2661 3073 8383 63	4803 9067 9335 2	3509 8658 6024 14	1149 8117 1039 68	4256 9291 0092 15	5502 8178 9101 31	4467 1903 1581 60	2717 6514 3190 28	2098 0403 9912 9	4703 4589 0424 61	0	0	0	1342 088
P	3160 1915 5872 66	5456 3501 8965 20	5967 5484 5386 94	4236 3489 9802 34	1315 2762 2196 86	3466 6974 1416 29	6084 6132 3933 03	1340 2172 9181 94	3839 4088 1275 59	1921 5903 4215 55	5327 0110 2312 52	2604 0254 5824 92	2043 7444 1615 15	3769 3150 7967 82
R	4090 6989 4092 11	1845 4204 6721 97	4629 2876 0842 77	9436 9151 7185 8	6268 7354 2167 52	3727 4555 1004 88	4940 8028 6863 80	4690 3380 3768 61	3752 1887 9472 47	9836 4076 6735	2114 7399 0242 08	1978 6381 7789 38	3323 4525 1236 58	2545 8545 3942 58
S	5985 9664	3851 7224	5028 6516	7317 0483	3167 1270	4568 4368	5237 9864	4619 2502	3428 2951	4486 8419	3445 3443	5919 1933	1068 2345	2414 4941

	9499 8	9165 23	6949 95	5148 9	8342 34	0001 40	7545 55	2401 52	1037 20	1282 79	6640 61	3325 19	2513 80	0128 67
T	3808 7785 7436 46	3781 4521 1254 71	3670 8359 4867 14	1039 5318 8327 80	3432 0328 8528 20	5156 4598 6439 69	3957 3685 9537 69	1339 7111 0628 89	1447 8832 5924 15	1873 4006 9117 27	1711 4289 7094 01	3311 5754 4295 50	1527 1993 6448 76	3142 4149 8505 19
Y	9044 1018 8916	7943 8536 6621	4974 4134 4627 20	5362 6604 0927 51	4734 0634 6963 28	2885 1726 6279 35	5080 0861 3270 15	6127 8402 4525 16	5163 4387 5755 13	4443 0822 9813 63	2265 8573 8300 44	2601 4090 2085 83	1830 5596 4342 01	6075 9518 5414 28
H U B	1440 2239 6218 72	2836 3589 8478 4	3668 2241 3601 13	1171 5608 1406 57	3650 4176 7122 85	6148 4664 1998 55	5435 9574 1708 55	4388 7217 6701 47	4795 7489 4478 74	1547 5510 0173 73	4891 5437 0259 32	3255 7245 8684 71	6195 7164 3517 14	2033 1425 0450 96

Decryption verification

- Decryption exactly recovered every diagonal entry (packed values and the seed)
- Full plaintext reconstructed by placing each recovered letter at every recovered position: "CRYPTOGRAPHY"

SECURITY DISCUSSION

Key-generation and modulus design

Key generation relies on a cryptographic construction rather than a general-purpose pseudorandom generator: every K_i and LT_2 are derived via HMAC-SHA256 counter-mode expansion (Section 3.1), avoiding the state-reconstruction weakness inherent to generators such as Mersenne Twister. The modulus q is fixed once ($q_{\max}$, Section 3.3) rather than derived per message, which closes a message-length side channel: both worked examples share the identical $q_{\max} = 62,728,147,919,399$ despite differing in length, confirmed computationally. LT_2 is likewise an HMAC-derived invertible lower-triangular matrix indistinguishable from random, rather than a matrix with fixed, guessable structure.

Limitations

The scheme's key relationships remain linear-algebraic in structure, it does not constitute a security proof against a capable cryptanalyst with structured (chosen-plaintext) access. The positional-packing base B and repeat capacity R (Section 3.2) are public parameters that bound the longest message and most-repeated letter the scheme can represent; messages exceeding these bounds are not supported without renegotiating B and R as part of Stage A. Finally, this paper's reference implementation is written for exposition and verification in a computer algebra system (Python/sympy) rather than as a hardened, side-channel-resistant production library; a real deployment would additionally need constant-time modular arithmetic to avoid timing side channels, which ordinary matrix arithmetic libraries do not guarantee.

CONCLUSION

We have presented a five-layer symmetric cipher combining a provably coprime, seed-parameterised Fibonacci labeling of a wheel graph, a genuine minimum-spanning-tree reduction, a dominator-coloring key partition proved correct for all wheel graphs (Theorem 1), and a fixed, alphabet-wide graph and key structure that removes any plaintext-dependence of the graph itself. Both worked examples, MATHEMATICS and

CRYPTOGRAPHY, share the identical fixed graph, spanning tree, key blocks, and permutation, and were verified computationally to decrypt exactly, recovering each full plaintext message, including every repeated letter, from the message-specific positional diagonal alone. We have stated the scheme's remaining limitations plainly, as concrete directions for further work.

REFERENCES

1. Bresar, B., Klavzar, S., & Rall, D. F. (2006). Dominator colorings of graphs. Manuscript, foundational reference for dominator coloring.
2. Koshy, T. (2001). Fibonacci and Lucas numbers with applications. Wiley.
3. Nayana, M. V., & Sobha, K. R. (2023). Encoding and decoding process using prime graceful labeling. Tuijin Jishu/Journal of Propulsion Technology, 44(4), 2638-2644.
4. Panma, S., & Rochanakul, P. (2021). Prime-graceful graphs. Thai Journal of Mathematics, 19(4), 1685-1697.
5. Rosa, A. (1967). On certain valuations of the vertices of a graph. In Theory of Graphs (International Symposium, Rome, 1966) (pp. 349-355). Gordon and Breach.
6. Selvarajan, T. M., & Subramoniam, R. (2018). Prime graceful labeling. International Journal of Engineering and Technology, 7(4.36), 750-752.