The IJICS (International Journal of Informatics and Computer Science)
Vol 10 No 2, July 2026, Page 142-152
ISSN 2548-8384 (online), ISSN 2548-8449 (print)
Available Online at https://ejurnal.stmik-budidarma.ac.id/index.php/ijics/index
DOI 10.30865/ijics.v10i2.9845

Performance Analysis of RSARC4 Hybrid Encryption Using the EM2B Key Generator

Elwin Yunith Purba, Aji Prabowo, Jesman Ritonga

Institut Bisnis Dan Komputer (IBK), Medan, Indonesia
Email: [1,*]elwinpurba.manorsa@gmail.com, [2]ajipr4bowo@gmail.com, [3,]jesmanritonga17@gmail.com
(* : elwinpurba.manorsa@gmail.co)
Submitted: **28/06/2026**; Accepted: **15/07/2026**; Published: **29/07/2026**

Abstract−This study evaluates the computational performance of a hybrid encryption scheme that combines RSA for key protection, RC4 for file encryption, and the EM2B mechanism for deterministic key expansion. The prototype was implemented as a PHP-based application and tested on 35 SQL text files. RC4 was used as the baseline, while the hybrid scheme was assessed using ciphertext size, character count, encryption time, decryption time, and restoration success. The reported averages show that RC4 required 1.33 s for encryption and 1.41 s for decryption, whereas RSAEM2BRC4 required 5.30 s and 100.09 s, respectively. The hybrid scheme also produced approximately 4.29 times greater ciphertext storage than RC4 alone. All files were restored to the same reported size after decryption; however, byte-level or hash-based integrity verification was not performed. Therefore, the results demonstrate functional implementation and quantify computational overhead, but they do not establish superior cryptographic security. RC4 is retained only as an experimental legacy baseline because current standards deprecate its operational use. Future evaluation should include authenticated encryption, entropy and randomness testing, key-sensitivity analysis, repeated timing trials, and cryptographic hash verification of restored files.

Keywords: Cryptography; RSA; RC4; EM2B; Key Generator

1. INTRODUCTION

The growth of digital information systems has increased the volume of text files exchanged and stored through networked applications. These files may contain operational records, database exports, or other information whose confidentiality and integrity must be protected. Cryptography provides mechanisms for transforming readable data into ciphertext and for managing the keys required to reverse that transformation. Current guidance emphasizes that algorithm selection, key length, key lifecycle, and implementation controls must be considered together; an encryption algorithm cannot be judged solely from the appearance or length of its ciphertext [1].

RSA is a public-key algorithm commonly used for protecting or transporting small cryptographic values rather than encrypting large files directly. Secure RSA implementation requires standardized encoding and padding, such as RSAES-OAEP, together with an appropriate modulus size and protected private-key management [2]. Symmetric algorithms are generally more efficient for bulk data, which motivates hybrid designs: a symmetric key encrypts the file, while an asymmetric mechanism protects the symmetric key. This division of responsibility can provide practical performance, but only when each component and its composition follow accepted cryptographic requirements.

RC4 is a historical stream cipher that is computationally simple and was therefore widely adopted in earlier systems. Subsequent cryptanalysis identified statistical biases in its keystream, and contemporary Internet security guidance prohibits or deprecates RC4 in secure protocols [3], [4]. Consequently, RC4 should not be recommended for new operational systems. It remains useful here only as a legacy experimental baseline for examining the behavior and overhead of the proposed combination. A modern deployment should instead use authenticated encryption, such as AES-GCM or ChaCha20-Poly1305, because these schemes protect confidentiality and also detect unauthorized modification [5], [6].

The EM2B mechanism used in this study transforms a primary key through ASCII conversion, modular arithmetic, and recursive key expansion. Its intended function is to extend a short input key to the length needed by the RC4 process. However, deterministic expansion is not equivalent to cryptographically secure random generation. A secure key-generation process must identify its entropy source, estimate the amount of entropy available, and validate the construction against recognized requirements [7]. Statistical test suites may help identify non-random patterns, but successful statistical tests alone cannot prove cryptographic security [8]. Therefore, the EM2B mechanism is evaluated in this article as an implementation component rather than assumed to improve security.

Previous work on hybrid encryption has commonly combined asymmetric key protection with a faster symmetric data-encryption algorithm. The main benefit of such a design is operational separation between key distribution and bulk encryption. Its limitations include additional storage, processing overhead, implementation complexity, and the possibility that a weak symmetric component or predictable key generator undermines the entire construction. The security strength of the full system is bounded by its weakest component and by the way keys, nonces, and parameters are generated and stored [9]. For that reason, ciphertext expansion and longer processing time cannot be interpreted as evidence of stronger security.

The research gap addressed by this study is the absence of a transparent performance analysis of the specific RSAEM2BRC4 composition on SQL text files. The original EM2B proposal describes a collaborative key-transformation concept [10], but the performance cost and security limitations of inserting the mechanism into a hybrid file-encryption workflow require clearer evaluation. This study therefore asks: (1) can the combination encrypt and decrypt the selected SQL text files; (2) how much storage and processing overhead does it introduce compared with RC4 alone; and (3) what conclusions can legitimately be drawn from the available measurements?

The IJICS (International Journal of Informatics and Computer Science)
Vol 10 No 2, July 2026, Page 142-152
ISSN 2548-8384 (online), ISSN 2548-8449 (print)
Available Online at https://ejurnal.stmik-budidarma.ac.id/index.php/ijics/index
DOI 10.30865/ijics.v10i2.9845

The contribution of the article is threefold. First, it documents the processing sequence for RSA key protection, EM2B key expansion, and RC4 data transformation. Second, it reports comparative measurements for 35 files using consistent performance indicators. Third, it separates demonstrated functional and performance findings from untested security claims. The evaluation does not claim resistance to cryptanalysis because it does not include entropy estimation, standardized randomness testing, key-sensitivity experiments, chosen-plaintext or related-key analysis, or an authenticated-encryption baseline. This distinction improves the interpretability of the results and establishes a reproducible direction for subsequent work.

The expected practical benefit is a clearer understanding of the trade-off produced by this legacy hybrid construction. Developers can observe the additional computation and storage introduced by the combination, while researchers can identify the validation steps still required before any security claim is made. For operational use, the design should be migrated to a standardized authenticated-encryption interface [11], keys should be derived with a reviewed key-derivation function [12], and RSA-based key transport should follow current implementation recommendations [13].

2. RESEARCH METHODOLOGY

2.1 Research Stages

The study followed five stages: problem identification, specification of the hybrid workflow, implementation of the PHP prototype, comparative testing on 35 SQL text files, and interpretation of performance and security limitations. The workflow first generated or expanded the symmetric key with EM2B, used RC4 to transform file content, protected key material through the RSA component, and then reversed the sequence during decryption. RC4-only processing served as the baseline.

Cryptographic evaluation in this study distinguishes functionality, performance, and security. Functionality concerns whether encryption and decryption complete; performance concerns time and storage overhead; security requires separate evidence concerning key strength, randomness, attack resistance, and integrity protection. Only the first two dimensions are measured directly in the available experiment [1], [9].

The hybrid architecture follows the general principle of using asymmetric cryptography for key protection and symmetric cryptography for bulk data. RSA implementation should follow standardized padding and key-management requirements [2], [13]. RC4 is analyzed only as a legacy baseline and is not recommended for deployment [3], [4].

2.2 Test Data

The experiment used 35 SQL-formatted text files. The available records report file size, character count, encryption time, and decryption time for RC4 and for the RSAEM2BRC4 combination. The files were selected to represent different text-file sizes processed by the prototype. Their detailed source, sensitivity classification, and public availability were not recorded in the original experiment; this limits independent replication and is treated as a threat to external validity.

2.3 Algorithm Implementation

2.3.1 EM2B Key Expansion

The EM2B mechanism converts the primary key into decimal ASCII values, computes a secondary value through modulo 26, and generates an expanded key byte through addition followed by modulo 256. When the expanded key remains shorter than the plaintext, successive characters are produced recursively from the two preceding key values. Equations (1)(3) and Figs. 12 describe the process. Because the construction is deterministic and the original experiment did not identify an external entropy source, it is described as key expansion rather than a validated random-key generator [7], [8], [10].

$$K_j = K_i \bmod 26 \quad (1)$$

$$K_{i[new]} = (K_i + K_J) \bmod 256 \quad (2)$$

Explanation:

a. K_i = Main Key,
b. K_j = Main Key do modulation with mod 26 (K_i mod 26),
c. $K_{i[new]}$ = New Key generated.

The scheme to explain the *EM2B* Generator Key and Increment Key process can be seen in the following image:

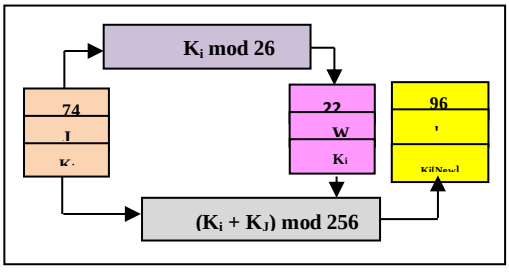


Fig 1. Process of the EM2B Key Algorithm (Source: E Mendrova E Manorsa and B Yako 2017)

And for the Icreament Key algorithm process: $IncK_i = (K_{i[max]} + K_{i[max-1]}) \bmod 256$ (3)

Explanation:

The IJICS (International Journal of Informatics and Computer Science)
Vol 10 No 2, July 2026, Page 142-152
ISSN 2548-8384 (online), ISSN 2548-8449 (print)
Available Online at https://ejurnal.stmik-budidarma.ac.id/index.php/ijics/index
DOI 10.30865/ijics.v10i2.9845

$IncK_i$ = Increament Key,
$K_{i[max]}$ = The last Key index in ASCII,
$K_{i[max-1]}$ = Previous Key Index in the key

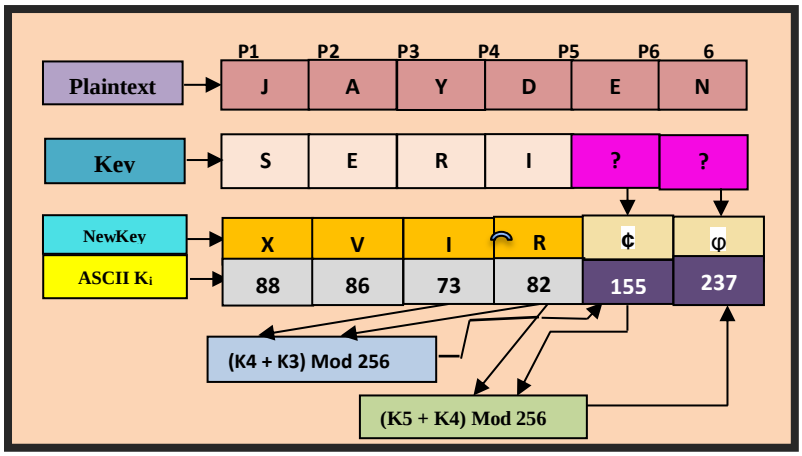


Fig 2. Increment Key Process (Source: E Mendrova E Manorsa and B Yako 2017)

The EM2B construction was derived from classical repeated-key and modular-transformation concepts described in the original proposal [10]. The implementation maps characters to integer values and extends the key until its length equals the plaintext length. This procedure is reproducible from Equations (1)(5), but its entropy, key space, and resistance to modern cryptanalytic attacks were not established by the available experiment.

$$E_i = (P_i + K_i) \ mod \ 26 \quad (4)$$

E_i, P_i and K_i are the encrypted result characters, message characters and key characters. To restore the ciphertext from the encryption results, the decryption process is carried out using the following equation:

$$D_i = (C_i - K_i) \ mod \ 26 \quad (5)$$

and D_i is the decrypted character, C_i is the ciphertext or password character, and K_i is the key character. An illustration of the use of the *EM2B* key generator algorithm can be seen in the following Figure:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	Plainteks (P)	E	l	w	i	n		Y	u	n	i	t	h		P	u	r	b	a
2	ASCII Code From Plaintext	69	108	119	105	110	32	89	117	110	105	116	104	32	80	117	114	98	97
3			=MOD(C6;26)				=(MOD(E6+E7;256)								=(MOD(L8+K8;256)				
4	EM2B Key and Increment Key																		
5	Kunci (K)	v	c	9	x	E	A	F	1	D	4	G							
6	ASCII Code Of Key	118	116	57	120	66	65	70	49	68	52	71							
7	Ki MOD 26	14	12	5	16	14	13	18	23	16	0	19	?	?	?	?	?	?	?
8	K = (Ki [max] + Ki[max] - 1) mod 256	132	128	62	136	80	78	88	72	84	52	90	142	232	118	94	212	50	6
9	Increamnet Key	„	€	>	ˆ	P	N	X	H	T	4	Z	Ž	è	v	^	Ô	2	ACK
10																			
11	Key to Binary Conversion																		
12	DEC2BIN(K)	10000100	10000000	111110	10001000	1010000	1001110	1011000	1001000	1010100	110100	1011010	10001110	11101000	1110110	1011110	11010100	110010	110
13			=DEC2BIN(B8)																
14	Encryption With EM2B Keys																		
15	(P + K) MOD 256	201	236	181	241	190	110	177	189	194	157	206	246	8	198	211	70	148	103
16	Ciperteks	É	ì	µ	ñ	¾	N	±	½	Â	OSC	Î	ö	è	Æ	Ó	F	"	g
17			=MOD(B2+B8;256)																
18	Convert Ciphertext to Binary																		
19	DEC2BIN(C)	11001001	11101100	10110101	11110001	10111110	1101110	10110001	10111101	11000010	10011101	11001110	11110110	1000	11000110	11010011	1000110	10010100	1100111
20			=MOD(B2+B8;256)																
21	Decryption With EM2B Key																		
22	(P + K) MOD 256	201	236	181	241	190	110	177	189	194	157	206	246	8	198	211	70	148	103
23	Ciperteks	É	ì	µ	ñ	¾	N	±	½	Â	OSC	Î	ö	è	Æ	Ó	F	"	g
24	(C - K) MOD 256	69	108	119	105	110	32	89	117	110	105	116	104	32	80	117	114	98	97
25	Plainteks	E	l	w	i	n		Y	u	n	i	t	h		P	u	r	b	a
26			=MOD(B15-B8;256)										=DEC2BIN(L24)						
27	Plaintext to Binary Conversion																		
28	DEC2BIN(P)	1000101	1101100	1110111	1101001	1101110	100000	1011001	1110101	1101110	1101001	1110100	1101000	100000	1010000	1110101	1110010	1100010	1100001

Fig 3. Table *EM2B* Key Generator Algorithm

The IJICS (International Journal of Informatics and Computer Science)
Vol 10 No 2, July 2026, Page 142-152
ISSN 2548-8384 (online), ISSN 2548-8449 (print)
Available Online at https://ejurnal.stmik-budidarma.ac.id/index.php/ijics/index
DOI 10.30865/ijics.v10i2.9845

When designing a cryptographic algorithm, maximum accuracy is essential. The level of security is key to the success of the cryptographic algorithm itself. Time efficiency also needs to be considered because if the encryption and decryption process takes a long time, it will be detrimental to encrypting data on a large scale.[5]. In general, the encryption and decryption process in the implementation of the RSA algorithm and the RC4 algorithm with the *EM2B* key generator in encrypting data can be seen in the following blog diagram:

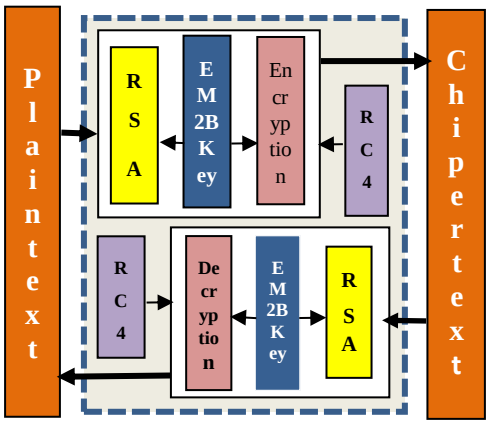


Fig 4. Encryption and Decryption Process With RSA + *EM2B* + RC4

The implemented workflow uses RC4 to transform the file content, EM2B to expand the input key, and RSA to protect key-related values. Figure 3 shows encryption and decryption as inverse processing paths. The experimental report does not retain the PHP version, processor, memory, operating system, RSA modulus length, RC4 key length, warm-up policy, or number of repeated timing trials. These omissions are explicitly acknowledged because they restrict reproducibility and prevent inferential timing analysis.

The analysis of the RSA algorithm can be seen as follows:

a. Randomly select two large and different prime numbers, p and q, but their size or the number of digits in the number base used must be the same.
b. Calculate the modulus n and the Euler's Tational function φn using the formula: n = pq
   φ(n) = (p−1)[q−1] where:
   n modulus (public key)
   p and q = Two randomly generated prime numbers.
c. Choose an integer e such that I < e < φ(n) and
   gcd, φn = 1, where:
   I = Integer,
   e = Public Key (Encryption Key),
   gcd = Greatest Common Divisor.
d. Calculate the value of an integer d where I < d < φ(n) as:
   d = e−1 mod φn or ed = I mod φn, where:
   d = Private Key (Decryption Key).
e. Create a table to represent each character.
f. Encrypt the plaintext (encrypted text) with the numbers corresponding to the table generated by process e, resulting in M, a set of numbers from the plaintext. This set of numbers is then block-blocked every four digits to form m1, m2, m3, ... mn. The encryption process is performed block by block, and each block's encryption formula is: c1 = m1e (mod n), c2 = m2e mod n, ... etc., resulting in a value of c, where c is the set of numbers c1, c2, c3, ... cn.
g. The decryption process is performed using the same logic as step F, performing the inverse calculation: m1 = c1d (mod n), m2 = c2d (mod n), ..., and so on, resulting in a value of M, where M = m1, m2, m3. The final value of M is presented in a table created as in step E above.

To enhance the security of the RSA algorithm, a security key is defined in the form of a private key password, a public key password, and a modulo generated from two prime numbers. This key will be continuously used by the sender and recipient of data to encrypt and decrypt data. If the system owner deems the security key password insecure, both parties will immediately notify them to change it completely.

Analysis of *EM2B* algorithm as follows:

a. Specify some words used as the primary key for encrypting data. Key is given a symbol with K_i where K_i= K_1, K_2 , K_3, ... K_n .
b. The key is converted into decimal ASCII numbers.
c. Determine the modulus value of 26 of each key character that has been converted into decimal places, K j= Ki mod 26.
d. Add K_i with K_j then modulate with 256 and generate a new key (K_{inew}) which is converted in decimal ASCII characters.
   In the EM2B Key algorithm, the key we choose does not have to have the same character length as plaintext. Plaintext may consist of several sentences and even paragraphs. The key will adjust the length of its character

The IJICS (International Journal of Informatics and Computer Science)
Vol 10 No 2, July 2026, Page 142-152
ISSN 2548-8384 (online), ISSN 2548-8449 (print)
Available Online at https://ejurnal.stmik-budidarma.ac.id/index.php/ijics/index
DOI 10.30865/ijics.v10i2.9845

with plaintext by using the increment key algorithm already stored in it. The performance analysis of the increment key algorithm can be seen below:

1. The maximum character index is summed with the previous character index (K_{imax}+K_{imax}−1), and generates a new key character index (K_{inew}).
2. The new key index (K_{inew}) becomes the maximum key index, then added again to the previous key index.
3. This looping step will stop if the maximum index of the key is equal to the plaintext maximum index $K_{imax} = P_{imax}$

2.4 Evaluation Method

The comparative evaluation used RC4 as the baseline and RSAEM2BRC4 as the treatment. The recorded measures were plaintext file size, ciphertext file size, character count, encryption time, decryption time, and whether the decryption process returned an output with the same reported size. Ratios were calculated from the reported averages to quantify overhead. No byte-by-byte comparison, cryptographic hash verification, entropy test, standardized randomness test, key-sensitivity test, or attack simulation was recorded; therefore, the analysis does not treat file-size equality or ciphertext expansion as proof of security or integrity.

The timing values are descriptive because the original record does not specify repeated trials, dispersion, confidence intervals, or a controlled warm-up procedure. Accordingly, no statistical significance test is reported. Future experiments should execute each file multiple times under a documented hardware and software configuration, report mean, median, standard deviation, and confidence intervals, and validate restored content with a cryptographic hash.

3. RESULT AND DISCUSSION

3.1 Manual Algorithm Verification

Analysis of RC4 algorithm as follows:
The ciphertext decryption process using the RC4 algorithm is the same as the key-scheduling process. To obtain the plaintext, the obtained ciphertext is XORed with the previously obtained pseudo-random bytes, resulting in the plaintext, or original text.
Messages are sent in ciphertext so that once they reach the recipient, they can be converted back to plaintext by XORing them with the same key. The processing of the message after it reaches the recipient is shown below.
The XORing process for pseudo-random bytes with ciphertext during decryption is as follows:

The XORing process for pseudo-random bytes with ciphertext during decryption is as follows:

Ciperteks	XOR	Pseudo random byte	Plaintext	ASCII Code Letters
00110000	⊕	0000010	00110010	50 = **2**
00110000	⊕	0000011	00110011	51 = **3**
00110101	⊕	0000001	00110100	52 = **4**
00111000	⊕	0000001	00111001	57 = **9**
00110101	⊕	0000011	00110110	54 = **6**
00110000	⊕	0000001	00110001	49 = **1**
00110100	⊕	0000010	00110110	54 = **6**
00110101	⊕	0000010	00110111	55 = **7**
00111010	⊕	00000011	00111001	57 = **9**
00110010	⊕	0000000	00110010	50 = **2**
00110010	⊕	0000001	00110011	51 = **3**
00110010	⊕	0000001	00110011	51 = **3**
00110011	⊕	0000000	00110011	51 = **3**
00110110	⊕	0000011	00110101	53 = **5**
00110000	⊕	0000001	00110001	49 = **1**
00110100	⊕	0000010	00110110	54 = **6**
00110000	⊕	0000000	00110000	48 = **0**
00111011	⊕	0000010	00111001	57 = **9**

The final plaintext result of RC4 algorithm decryption is: **2349 616 792 3335 1609**

Next, the plaintext from the RC4 algorithm decryption results above is then decrypted using the secret key in the RSA algorithm as follows: **SK = 1019**
Ciphertext blocks are decrypted as follows:

2349^{1019} mod 3337 = 657 = x1
616^{1019} mod 3337 = 669 = x2
792^{1019} mod 3337 = 837 = x3
3335^{019} mod 3337 = 765 = x4
1609^{1019} mod 3337 = 78 = x5

The IJICS (International Journal of Informatics and Computer Science)
Vol 10 No 2, July 2026, Page 142-152
ISSN 2548-8384 (online), ISSN 2548-8449 (print)
Available Online at https://ejurnal.stmik-budidarma.ac.id/index.php/ijics/index
DOI 10.30865/ijics.v10i2.9845

The other plaintext blocks are returned in a similar manner. Finally, we recover the original plaintext, which is:

P = 65766983776578

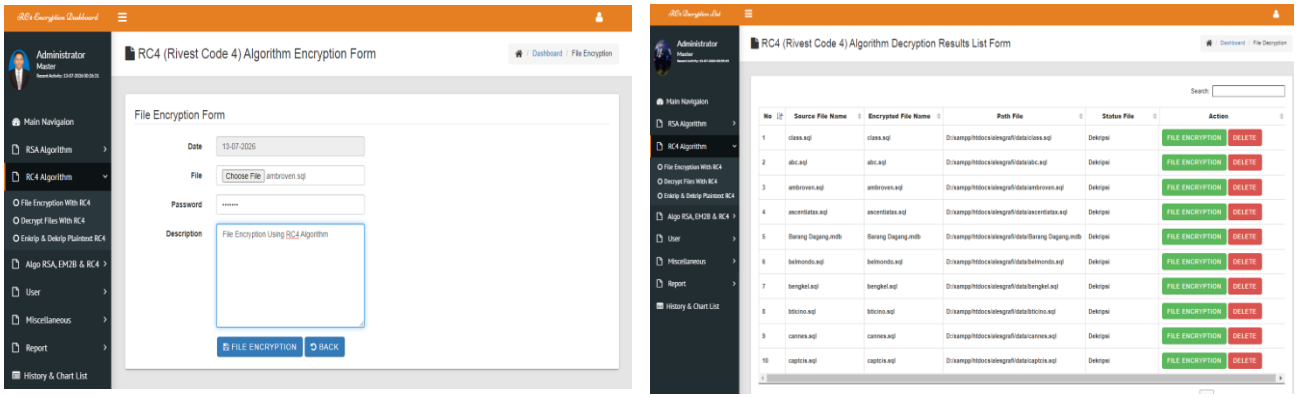
The decimal plaintext above is converted back into ASCII characters as follows:

65 = A 77 = M
76 = L 65 = A
69 = E 78 = N
83 = S

And the result of the collaboration/combination between the RSA and RC4 algorithms, the final plaintext decryption result is: **P = ALESMAN.**

The prototype was evaluated on 35 SQL-formatted text files. RC4-only processing and the RSAEM2BRC4 workflow were applied to the same file set. The analysis focuses on functional completion, storage expansion, character-count expansion, and processing time. Because the original dataset documentation is incomplete, these results characterize the tested sample only and should not be generalized to other file types or production environments.

3.2 RCA Baseline Implementasion



(a) Encryption Process With RC4 Algorithm **(b)** Decryption Process With RC4 Algorithm

Fig 5. (a)Encryption Process With RC4 Algorithm, **(b)**Decryption Process With RC4 Algorithm

3.3 RC4 Performance Results

Figures 5 show the RC4 encryption and decryption interfaces used for the baseline test. Table 1 reports the file-level results for the 35 SQL text files. Across the sample, the reported average ciphertext size was 1,932 KB, while average encryption and decryption times were 1.33 s and 1.41 s. The close encryption and decryption times are consistent with the symmetric nature of a stream-cipher workflow, although the absence of repeated trials means that run-to-run variability cannot be quantified.

Table 1. RC4 Algorithm Encryption and Decryption Results

File Name	Algorithm	Number of Characters After Encryption	Number of Chars Sth Descriptio	Initial File Size (Kb)	Files Size After Encryptio n	Files Size After Decryption (Kb)	Encrypti on Time (second)	Description Time (second)
abc.sql	RC4	662,859	223,311	218	647	218	0.38	0.44
ambroven.sql		3,573,749	1,195,948	1,168	3,490	1,168	2.18	2.32
asciientax.sql		298,361	100,675	98	291	98	0.18	0.18
Barang Dagang.mdb		647,415	280,576	274	632	274	0.46	0.53
belmondo.sql		210,866	71,276	70	206	70	0.14	0.13
bengkel.sql		614,716	207,278	202	600	202	0.34	0.37
bticino.sql		644,354	217,324	212	629	212	0.39	0.4
cannes.sql		1,575,042	527,015	515	1,538	515	0.94	1.01
captcis.sql		697,148	233,741	228	681	228	0.43	0.41
data.mdb		3,360,343	1,511,424	1,476	3,282	1,476	2.72	2.97
cityfingersql		638,381	215,193	210	623	210	0.36	0.42
cp.sql		65,223	220,207	215	637	215	0.38	0.39
DataAbsen.mdb		1,296,720	577,536	564	1,266	564	1.15	1.09
DB_Hotel.mdb		2,700,572	1,212,416	1,184	2,637	1,184	2.18	2.29
dbMaster.mdb		1,524,129	655,360	640	1,488	640	1.12	1.18
email.sql		2,444,176	819,864	801	2,387	801	1.48	1.55

The IJICS (International Journal of Informatics and Computer Science)
Vol 10 No 2, July 2026, Page 142-152
ISSN 2548-8384 (online), ISSN 2548-8449 (print)
Available Online at https://ejurnal.stmik-budidarma.ac.id/index.php/ijics/index
DOI 10.30865/ijics.v10i2.9845

File Name	Algorithm	Number of Characters After Encryption	Number of Chars Sth Descriptio	Initial File Size (Kb)	Files Size After Encryption	Files Size After Decryption (Kb)	Encryption Time (second)	Description Time (second)
dbRupali.mdb		3,218,498	1,380,352	1,348	3,143	1,348	2.51	2.65
Employee.mdb		1,547,246	626,688	612	1,511	612	1.06	1.19
Hotel.mdb		2,694,011	1,212,416	1,184	2,631	1,184	2.06	2.26
invoice.sql		2,959,478	991,328	968	2,890	968	1.72	1.9
stem.mdb		2,790,055	1,216,512	1,188	2,725	1,188	2.11	2.26
Info.mdb		718,929	319,488	312	702	312	0.58	0.62
v.sql		7,629,513	2,562,987	2,503	7,451	2,503	4.69	4.81
1s1.sql		2,810,962	942,104	920	2,745	920	1.6	1.89
mbpointapp.sql		442,759	149,092	146	432	146	0.28	0.3
Multi.mdb		295,566	118,784	116	289	116	0.2	0.22
posdb.mdb		846,433	356,352	348	827	348	0.64	0.69
Prabayar.mdb		1,310,211	593,920	580	1,280	580	1.07	1.13
rezekimakmur.sql		3,106,690	1,042,646	1,018	3,034	1,018	1.87	1.91
RSI.mdb		1,971,653	872,448	852	1,925	852	1.49	1.58
SecurePark.mdb		1,402,705	643,072	628	1,370	628	1.11	1.18
Siswa Baru.mdb		352,389	133,120	130	344	130	0.22	0.25
snapvan.sql		8,358,648	2,800,075	2,734	8,163	2,734	5.11	5.32
Stock.mdb		954,351	405,504	396	932	396	0.72	0.73
sumosave.sql		4,276,674	1,430,740	1,397	4,176	1,397	2.64	2.73
AVERAGE		**1,961,167**	**744,765**	**727**	**1,932**	**727**	**1.33**	**1.41**

The following is a graphic display of the encryption results from the RC4 algorithm in the form of the number of characters, file size and the time required for each file after being encrypted and decrypted with the RC4 algorithm.

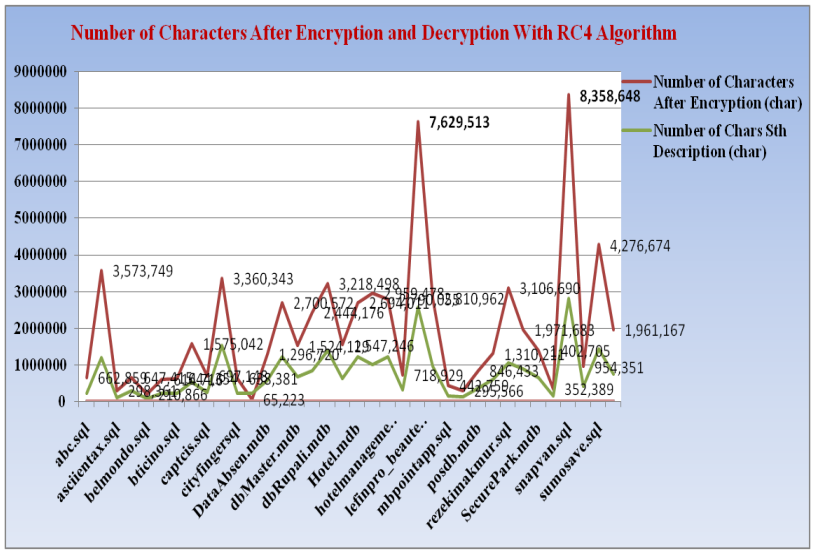


Fig 6. Graphic Display of Number of Characters With RC4 Algorithm

3.4 RSAEM2BRC4 Performance Results

Table 2 and Figure 6 compare the RC4 baseline with the RSAEM2BRC4 workflow. The hybrid method produced an average ciphertext size of 8,285 KB compared with 1,932 KB for RC4, corresponding to approximately 4.29 times the storage requirement. Average character count increased from 1,961,167 to 7,912,310, or approximately 4.03 times. These ratios quantify representation and key-processing overhead; they do not demonstrate additional security.

The IJICS (International Journal of Informatics and Computer Science)
Vol 10 No 2, July 2026, Page 142-152
ISSN 2548-8384 (online), ISSN 2548-8449 (print)
Available Online at https://ejurnal.stmik-budidarma.ac.id/index.php/ijics/index
DOI 10.30865/ijics.v10i2.9845

Table 3. RSA + EM2B + RC4 Character Count Comparison Versus RC4 Algorithm

File Name	RC4 ALGORITHM							RSA + EM2B + RC4 ALGORITHM								
	File Size Before Encryption (Kb)	Number of Characters After Encryption (char)	Number of Chars Sth Description (char)	File Size After Encryption (Kb)	File Size After Decryption (Kb)	Encryption Time (second)	Description Time (second)	P value	Q value	Number of Characters After Encryption (char)	Number of Chars Sth Description (char)	File Size After Encryption (Kb)	File Size After Decryption (Kb)	Encryption Time (second)	Description Time (second)	File Size After Decryption (Kb)
abc.sql	218	662,859	223,311	647	218	0.38	0.44	53	97	3,225,807	223,311	3,150	218	1.7	39.66	218
ambroven.sql	1,168	3,573,749	1,195,948	3,490	1,168	2.18	2.32	53	97	17,150,517	1,195,948	16,749	1,168	9.45	207.62	1,168
asciientaxsql	98	298,361	100,675	291	98	0.18	0.18	53	97	1,438,820	100,675	1,405	98	0.78	17.18	98
Barang Dagang.mdb	274	647,415	280,576	632	274	0.46	0.53	53	97	2,037,162	280,576	1,989	274	1.14	9.89	274
belmondo.sql	70	210,866	71,276	206	70	0.14	0.13	53	97	1,032,617	71,276	1,008	70	0.55	12.62	70
bengkel.sql	202	614,716	207,278	600	202	0.34	0.37	53	97	3,014,595	207,278	2,944	202	1.63	36.75	202
bticino.sql	212	644,354	217,324	629	212	0.39	0.4	53	97	3,169,239	217,324	3,095	212	1.75	38.45	212
cannes.sql	515	1,575,042	527,015	1,538	515	0.94	1.01	53	97	7,527,258	527,015	7,351	515	4.06	94.82	515
captcis.sql	228	697,148	233,741	681	228	0.43	0.41	53	97	3,360,330	233,741	3,282	228	2.12	47.01	228
data.mdb	1,476	3,360,343	1,511,424	3,282	1,476	2.72	2.97	53	97	9,568,051	1,511,424	9,344	1,476	5.26	24.91	1,476
cityfingersql	210	638,381	215,193	623	210	0.36	0.42	53	97	3,137,448	215,193	3,064	210	1.68	37.45	210
cp.sql	215	65,223	220,207	637	215	0.38	0.39	53	97	3,206,144	220,207	3,131	215	1.68	39.07	215
DataAbsen.mdb	564	1,296,720	577,536	1,266	564	1.15	1.09	53	97	4,349,131	577,536	4,247	564	2.32	22.23	564
email.sql	1,184	2,444,176	819,864	2,387	801	1.48	1.55	53	97	1,809,741	819,864	11,533	801	6.36	138.57	1,184
dbRupali.mdb	640	3,218,498	1,380,352	3,143	1,348	2.51	2.65	53	97	2,206,751	1,380,352	11,921	1,348	6.06	89.24	640
Employee.mdb	801	1,547,246	626,688	1,511	612	1.06	1.19	53	97	6,300,522	626,688	6,153	612	3.96	53.58	801
Hotel.mdb	1,348	2,694,011	1,212,416	2,631	1,184	2.06	2.26	53	97	9,509,567	1,212,416	9,287	1,184	5.2	58.55	1,348
invoice.sql	612	2,959,478	991,328	2,890	968	1.72	1.9	53	97	14,167,275	991,328	13,835	968	8.27	208.87	612
hotelmanagement.mdb	1,184	2,790,055	1,216,512	2,725	1,188	2.11	2.26	53	97	10,268,421	1,216,512	10,028	1,188	7.46	85.07	1,184
Info.mdb	968	718,929	319,488	702	312	0.58	0.62	53	97	2,065,397	319,488	2,017	312	1.32	7.19	968
lefinpro_kev.sql	1,188	7,629,513	2,562,987	7,451	2,503	4.69	4.81	53	97	37,333,498	2,562,987	36,459	2,503	25.25	549.62	1,188
ls1.sql	312	2,810,962	942,104	2,745	920	1.6	1.89	53	97	13,633,602	942,104	13,314	920	8.41	230.91	312
mbpointapp.sql	2,503	442,759	149,092	432	146	0.28	0.3	53	97	2,156,343	149,092	2,106	146	1.32	30.61	2,503
Multi.mdb	920	295,566	118,784	289	116	0.2	0.22	53	97	856,075	118,784	836	116	0.47	5.29	920
posdb.mdb	146	846,433	356,352	827	348	0.64	0.69	53	97	2,935,625	356,352	2,867	348	1.94	24.58	146
Prabayar.mdb	116	1,310,211	593,920	1,280	580	1.07	1.13	53	97	4,229,834	593,920	4,131	580	2.77	20.21	116
rezekimakmur.sql	348	3,106,690	1,042,646	3,034	1,018	1.87	1.91	53	97	15,161,124	1,042,646	14,806	1,018	9.27	234.38	348
RSI.mdb	580	1,971,683	872,448	1,925	852	1.49	1.58	53	97	6,938,893	872,448	6,776	852	6.65	64.98	580
SecurePark.mdb	1,018	1,402,705	643,072	1,370	628	1.11	1.18	53	97	4,332,863	643,072	4,231	628	3.91	28.4	1,018
Siswa Baru.mdb	852	352,389	133,120	344	130	0.22	0.25	53	97	1,285,169	133,120	1,255	130	0.74	14.49	852
snapvan.sql	628	8,358,648	2,800,075	8,163	2,734	5.11	5.32	53	97	40,524,463	2,800,075	39,575	2,734	27.27	604.88	628
Stock.mdb	130	954,351	405,504	932	396	0.72	0.73	53	97	3,366,021	405,504	3,287	396	2.61	33.51	130
sumosave.sql	2,734	4,276,674	1,430,740	4,176	1,397	2.64	2.73	53	97	20,729,732	1,430,740	20,244	1,397	13.51	288.82	2,734
DB_Hotel.mdb	396	2,700,572	1,212,416	2,637	1,184	2.18	2.29	53	97	9,490,582	1,212,416	9,268	1,184	5.27	63.04	396
dbMaster.mdb	1,397	1,524,129	655,360	1,488	640	1.12	1.18	53	97	5,412,244	655,360	5,285	640	2.89	40.57	1,397
RATA-RATA	**727**	**1,961,167**	**744,765**	**1,932**	**727**	**1.33**	**1.41**	**53**	**97**	**7,912,310**	**744,765**	**8,285**	**727**	**5.3**	**100.09**	**727**

The IJICS (International Journal of Informatics and Computer Science)
Vol 10 No 2, July 2026, Page 142-152
ISSN 2548-8384 (online), ISSN 2548-8449 (print)
Available Online at https://ejurnal.stmik-budidarma.ac.id/index.php/ijics/index
DOI 10.30865/ijics.v10i2.9845

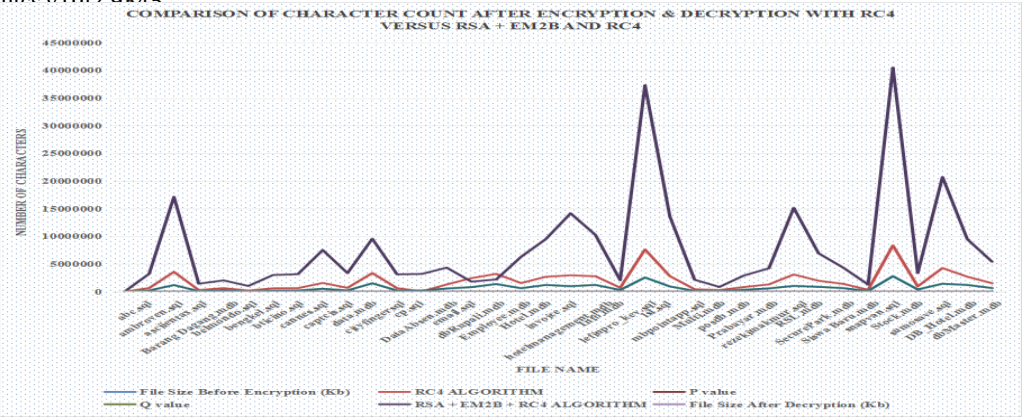


Fig 7. Comparison Chart of the Number of Characters between the RC4 Algorithm and the Combined RSA + EM2B and RC4 Algorithms

3.5 Comparative Performance Analysis

The encryption-time average increased from 1.33 s for RC4 to 5.30 s for the hybrid workflow. On the reported averages, the combined process was approximately 3.98 times slower during encryption. The difference is expected because the hybrid path performs additional key expansion and RSA-related operations beyond the symmetric transformation. This result is practically important: additional cryptographic components have measurable cost, and their inclusion must be justified by a security property that is independently evaluated.

The decryption-time difference was substantially larger. RC4 required an average of 1.41 s, whereas the combined method required 100.09 s, a ratio of approximately 70.99. This asymmetric increase suggests that the implemented RSA or conversion stage dominates the reverse path. The available records do not isolate each component, so the precise bottleneck cannot be identified. Profiling should separately measure RSA private-key operations, EM2B expansion, format conversion, file input/output, and RC4 transformation.

The reported storage expansion also has operational consequences. A 4.29-fold average increase affects disk capacity, transfer time, backup duration, and memory use. Ciphertext length can grow because of encoding, block representation, metadata, or repeated conversion to textual digits. The implementation should therefore document the serialization format and distinguish raw ciphertext bytes from their hexadecimal, decimal, or Base64 representation. Without that distinction, character count is not a reliable measure of cryptographic complexity.

3.6 Functional Correctness and Integrity

All 35 test files were reported as decryptable, and the average size before encryption and after decryption was 727 KB. This supports functional reversibility at the file-size level. Nevertheless, equal size does not guarantee identical content: two files can contain different bytes while having the same length. The experiment should calculate SHA-256 or another approved hash on each plaintext before encryption and compare it with the hash of the restored plaintext after decryption. Byte-for-byte comparison should also be reported.

The current workflow provides confidentiality transformation but does not report an authentication tag or message authentication code. Consequently, the experiment does not demonstrate detection of deliberate ciphertext modification. Modern authenticated-encryption schemes bind confidentiality and integrity through a verified tag [5], [6], [11]. A revised prototype should include AES-GCM or ChaCha20-Poly1305 as a contemporary baseline and reject altered ciphertext before plaintext is released.

3.7 Security Interpretation

RC4 has known keystream biases and is deprecated in contemporary security protocols [3], [4]. Combining it with RSA does not remove those biases because RSA protects key material rather than changing the statistical behavior of the RC4 keystream. The experimental findings must therefore be interpreted as a legacy performance study, not as evidence that the construction is suitable for current secure deployment.

The EM2B mechanism expands key material deterministically. The original evaluation does not estimate input entropy, min-entropy, output unpredictability, collision behavior, or sensitivity to related keys. NIST guidance requires explicit treatment of entropy sources for random-bit generation [7]. Randomness suites can be used as diagnostic tools [8], but cryptographic review must also consider whether an attacker who knows part of the key or the recurrence can predict subsequent bytes.

Ciphertext expansion and longer processing time are not security metrics. A secure algorithm may produce ciphertext only slightly longer than plaintext, while an insecure encoding can produce very long output. Security claims require a defined threat model, standardized algorithms, reviewed parameters, key-management controls, and evidence against relevant attacks. The present study measures performance overhead only; it does not establish that RSAEM2BRC4 is more secure than RC4 or modern authenticated encryption.

3.8 Comparison with Recommended Alternatives

For a modern hybrid system, RSA should be implemented with standardized padding and a suitable modulus, while the bulk data should be protected with authenticated encryption [2], [5]. AES-GCM provides confidentiality and authentication under a secret key and unique nonce. ChaCha20-Poly1305 provides a widely specified alternative for

The IJICS (International Journal of Informatics and Computer Science)
Vol 10 No 2, July 2026, Page 142-152
ISSN 2548-8384 (online), ISSN 2548-8449 (print)
Available Online at https://ejurnal.stmik-budidarma.ac.id/index.php/ijics/index
DOI 10.30865/ijics.v10i2.9845

software environments [6]. Both approaches directly address manipulation through an authentication tag, a property not evaluated in the current RC4 workflow.

The key-generation component should also be separated from password processing. If a user supplies a password, a password-based derivation or reviewed extraction-and-expansion mechanism should be used instead of assuming that recursive character operations create entropy [12]. If random keys are generated, the system should rely on a cryptographically secure random source validated under recognized requirements [7]. These changes would convert the prototype from a legacy algorithm experiment into a design aligned with current engineering practice.

3.9 Threats to Validity and Limitations

Internal validity is limited by the lack of documented repetitions, timing dispersion, and component-level profiling. Observed differences may include environmental noise, caching, file-system effects, and implementation overhead. Construct validity is limited because character count and ciphertext size were initially treated as indicators of security even though they measure representation overhead. The revised interpretation restricts those indicators to performance and storage.

External validity is limited to the 35 tested SQL text files. Their source, content type, sensitivity, and complete size distribution were not retained, and no binary or multilingual files were tested. Reproducibility is further limited by missing hardware, operating-system, PHP-version, key-length, and timing-procedure records. These details must be captured in a subsequent experiment before results can be generalized.

Security validity is limited by the absence of entropy estimation, standardized randomness analysis, key-sensitivity testing, cryptanalytic experiments, authenticated-integrity testing, and a modern baseline. The study therefore makes no claim that EM2B improves key quality or that the hybrid scheme resists practical attacks. Its defensible outcome is a functional prototype and a quantified description of computational and storage overhead.

Despite these limitations, the experiment provides a useful negative and engineering-oriented result: adding cryptographic stages creates substantial cost, particularly during decryption, and additional complexity must not be equated with additional security. The next evaluation should use documented parameters, repeated measurements, per-component profiling, hash-based recovery validation, authenticated-encryption baselines, and explicit attack scenarios.

A stronger follow-up design should pre-register the test protocol and publish anonymized file characteristics, source code, configuration files, and raw timing observations. Each algorithm should process identical inputs under isolated conditions, and throughput should be reported by file-size group. Repeated measurements would permit confidence intervals and a paired statistical comparison because every file is processed by both methods. This design would separate algorithmic cost from environmental variation and make the performance findings independently auditable.

Security evaluation should be conducted independently from performance evaluation. Recommended experiments include bit-level entropy summaries, standardized diagnostic randomness tests, key-sensitivity analysis, repeated-key and related-key scenarios, ciphertext-tampering tests, and recovery verification using cryptographic hashes. These tests would not automatically prove security, but they would provide evidence relevant to clearly defined claims. Until such evidence is available, the measured values should be interpreted solely as implementation behavior on the reported dataset.

4. CONCLUSION

This study implemented and evaluated a hybrid RSAEM2BRC4 workflow on 35 SQL text files and compared it with RC4-only processing. The recorded averages show that the hybrid method required approximately 4.29 times more ciphertext storage, 3.98 times longer encryption time, and 70.99 times longer decryption time than RC4. All tested files were restored to the same reported size, demonstrating functional reversibility at the size level. However, content identity was not verified byte-for-byte or with a cryptographic hash, so lossless recovery cannot be conclusively claimed. The results quantify computational and storage overhead but do not establish improved cryptographic security. RC4 is deprecated for operational use, ciphertext expansion is not a security metric, and the available experiment did not validate EM2B entropy, randomness, key sensitivity, or attack resistance. The proposed combination should therefore be treated as a legacy experimental prototype rather than a recommended security solution. Future work should document the complete environment, repeat each timing trial, report statistical dispersion, verify restored files with hashes, profile each processing component, and compare the implementation with authenticated encryption such as AES-GCM or ChaCha20-Poly1305. The principal implication is that additional algorithmic complexity must be justified through validated security properties, not inferred from ciphertext length or processing cost.

REFERENCES

[1] E. Barker, Recommendation for Key Management: Part 1—General, NIST Special Publication 800-57 Part 1 Rev. 5. Gaithersburg, MD, USA: National Institute of Standards and Technology, 2020, doi: 10.6028/NIST.SP.800-57pt1r5.
[2] K. Moriarty, B. Kaliski, J. Jonsson, and A. Rusch, "PKCS #1: RSA Cryptography Specifications Version 2.2," RFC 8017, Nov. 2016, doi: 10.17487/RFC8017.
[3] Y. Sheffer, P. Saint-Andre, and T. Fossati, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)," RFC 9325, Nov. 2022, doi: 10.17487/RFC9325.
[4] L. Velvindron, "Deprecating RC4 in Secure Shell (SSH)," RFC 8758, Apr. 2020, doi: 10.17487/RFC8758.
[5] M. Dworkin, Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC, NIST Special Publication 800-38D. Gaithersburg, MD, USA: National Institute of Standards and Technology, 2007, doi: 10.6028/NIST.SP.800-38D.
[6] Y. Nir and A. Langley, "ChaCha20 and Poly1305 for IETF Protocols," RFC 8439, Jun. 2018, doi: 10.17487/RFC8439.
[7] M. S. Turan et al., Recommendation for the Entropy Sources Used for Random Bit Generation, NIST Special Publication 800-90B. Gaithersburg, MD, USA: National Institute of Standards and Technology, 2018, doi: 10.6028/NIST.SP.800-90B.

The IJICS (International Journal of Informatics and Computer Science)
Vol 10 No 2, July 2026, Page 142-152
ISSN 2548-8384 (online), ISSN 2548-8449 (print)
Available Online at https://ejurnal.stmik-budidarma.ac.id/index.php/ijics/index
DOI 10.30865/ijics.v10i2.9845

[8] A. Rukhin et al., A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications, NIST Special Publication 800-22 Rev. 1a. Gaithersburg, MD, USA: National Institute of Standards and Technology, 2010, doi: 10.6028/NIST.SP.800-22r1a.
[9] E. Barker and A. Roginsky, Transitioning the Use of Cryptographic Algorithms and Key Lengths, NIST Special Publication 800-131A Rev. 2. Gaithersburg, MD, USA: National Institute of Standards and Technology, 2019, doi: 10.6028/NIST.SP.800-131Ar2.
[10] E. H. A. Mendrofa, E. Y. P. Manurung, B. Y. Sihombing, and R. W. Sembiring, "Collaborative encryption algorithm between Vigenère cipher, rotation of matrix, and one-time pad," in Proc. Int. Conf. Computer, Environment, Social Science, Health Science, Agriculture and Technology (ICEST), 2017.
[11] D. McGrew, "An Interface and Algorithms for Authenticated Encryption," RFC 5116, Jan. 2008, doi: 10.17487/RFC5116.
[12] H. Krawczyk and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)," RFC 5869, May 2010, doi: 10.17487/RFC5869.
[13] E. Barker, L. Chen, A. Roginsky, A. Vassilev, and R. Davis, Recommendation for Pair-Wise Key-Establishment Schemes Using Integer Factorization Cryptography, NIST Special Publication 800-56B Rev. 2. Gaithersburg, MD, USA: National Institute of Standards and Technology, 2019, doi: 10.6028/NIST.SP.800-56Br2.