ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26oct012

Evaluating Genetic-Algorithm Key Generation for OTP Encryption Against AEAD Standards

Ahmad Salah Aljalikh[1*]; Mohamad Mostafa Haajjouz[2*]

[1]Master Student, Web Science Program, Syrian Virtual University, Damascus, Syria
[2]Supervisor (Dr.), Faculty of Science, University of Homs, Homs, Syria

Corresponding Author: Ahmad Salah Aljalikh[1]; Mohamad Mostafa Haajjouz[2*]

Publication Date: 2026/11/02

Abstract: One-Time Pad (OTP) encryption offers perfect secrecy, but only if its key is truly random, as long as the message, and never reused. Prior studies proposed heuristic key generators for OTP, such as genetic algorithms (GA), linear congruential generators (LCG) and neural networks, usually without a unified comparison against the authenticated-encryption (AEAD) standards used in practice. This paper presents a reproducible benchmark of five methods: OTP-OS, LCG-XOR, GA-XOR, AES-GCM and ChaCha20-Poly1305. The main experiment contains 1,200 randomized observations over four message sizes (4 KiB to 5 MiB) with 30 replicated blocks, complemented by tamper-detection, keystream-diagnostic and GA-isolation experiments. AES-GCM was the fastest method from 64 KiB upward (6.90 ms at 5 MiB), whereas GA-XOR was 17.4 to 1,554.9 times slower than the fastest method depending on size. Each AEAD cipher rejected all 90 tampered packages, while each of the three XOR-based methods accepted all 90 modifications silently. LCG-XOR reached an ideal entropy of 8.000000 bit/byte yet repeated its keystream every 256 bytes. Finally, GA improved its own fitness function over a single seed but not over random search with an equal budget of 156 evaluations, nor on held-out data. Heuristic key generation therefore showed no demonstrated security benefit over standard AEAD.

Keywords: *One-Time Pad, Genetic Algorithm, AES-GCM, ChaCha20-Poly1305, Authenticated Encryption, Benchmarking.*

How to Cite: Ahmad Salah Aljalikh; Mohamad Mostafa Haajjouz (2026) Evaluating Genetic-Algorithm Key Generation for OTP Encryption Against AEAD Standards. *International Journal of Innovative Science and Research Technology*, 11(10), 2458-2461. https://doi.org/10.38124/ijisrt/26oct012

I. INTRODUCTION

Shannon proved that the One-Time Pad (OTP) is the only cipher with unconditional perfect secrecy, provided that the key is uniformly random, as long as the message, and used once [1], [2]. These conditions make key generation and distribution the practical bottleneck. A line of work therefore replaces the ideal key with the output of heuristic generators: genetic algorithms [3], [4], linear congruential generators [5] and deep neural networks [6], [7]. Such proposals, however, are rarely compared under a single protocol with the AEAD standards used in practice [8], namely AES-GCM [9] and ChaCha20-Poly1305 [10], which provide confidentiality and integrity together.

This paper addresses that gap. Its contributions are: (1) a randomized and reproducible benchmark engine that measures time, memory, entropy and functional correctness of five methods; (2) a statistical analysis with Holm-corrected paired tests and effect sizes; (3) tamper-detection and keystream-diagnostic experiments that separate descriptive randomness indicators from real security guarantees; and (4) an isolation experiment testing whether the GA adds value beyond equal-budget random search.

II. RELATED WORK

Sokouti et al. introduced a genetic random key generator for OTP [3], and Jain and Chaudhari proposed an improved GA for a deterministic OTP key generator [4]. Apdilah et al. compared linear and quadratic congruential generators for OTP keys [5]. Neural approaches include the 3D CUBE key-generation algorithm [6], and a recent review classifies AI-based key-generation trends [7]. On the standards side, Bellare and Namprempre formalized authenticated encryption [8], and AES-GCM and ChaCha20-Poly1305 are specified in [9] and [10]. Statistical test suites such as NIST SP 800-22 [11] assess randomness but do not prove unpredictability. Further GA-based key generators exist for stream and symmetric ciphers [12], [13], and OTP key management remains a separate problem [14]. The cited studies largely treat key generation and standard AEAD separately, and few combine tamper detection, equal-budget random-search controls and held-out evaluation, which motivates this benchmark.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26oct012

III. METHODOLOGY

➢ *Methods*

LCG-XOR XORs the message with the keystream of a linear congruential generator whose secret is an 8-byte state (low-order state byte used); with a power-of-two modulus the low-order byte has a very short period [15]. GA-XOR searches a 32-byte seed with a genetic algorithm (population 12, 12 generations, two elite individuals, crossover and single-bit mutation, i.e. 156 fitness evaluations) and expands the seed into a keystream with SHAKE-256. Fitness is computed on a 4,096-byte SHAKE-256 sample as $f = H/8 - 0.25|\rho| - 0.05 \cdot \min(|\chi^2 - 255|/255, 2)$, where H is the Shannon entropy (bit/byte), ρ the serial correlation and χ^2 the byte-uniformity statistic. The search PRNG seed is fixed, so evolved seeds are not secrets. OTP-OS XORs the message with a key of message length drawn from the operating-system CSPRNG (os.urandom); it is a cost baseline, not an unconditionally secret OTP. AES-GCM and ChaCha20-Poly1305 use a 32-byte key and a fresh 12-byte nonce per package, and add 28 bytes (nonce and 16-byte tag).

➢ *Experimental Design*

The main experiment crosses 5 methods, 4 message sizes (4 KiB, 64 KiB, 1 MiB, 5 MiB) and 2 measurement profiles over 30 replicated blocks, giving 1,200 observations. Within each block the order of sizes, methods and profiles is randomized, two warm-up runs precede recording, and inputs are deterministic SHAKE-256 streams. Time inference uses the timing-only profile, while the traced profile (tracemalloc) measures peak memory, because tracing inflated short-run times by up to 8.9 times. The primary variable is the full-cycle time (key generation, encryption and decryption), so that the GA search cost is not hidden. All runs used one x86 (Intel) machine, Python 3, NumPy and the cryptography package (minimum versions: NumPy 2.0, cryptography 46).

➢ *Statistics and Controls*

A Friedman test with Kendall's W gives the omnibus effect, followed by planned paired Wilcoxon contrasts with Holm correction. Effect sizes are geometric time ratios with bootstrap 95% confidence intervals and rank-biserial correlation. Integrity is tested by altering one byte at the start, middle or end of 30 packages per method (90 challenges). Keystream diagnostics report Shannon entropy, χ^2 and lag correlations. The GA is isolated by comparing it with random search of the same 156-evaluation budget and with a single seed, on a training sample and on an independent hold-out sample.

➢ *Platform*

The engine is also wrapped by a Flask API signed with HMAC-SHA256 and a PHP/MySQL research portal; a FLAN-T5-Base model [16] drafts explanations bound to the measured evidence by SHA-256 fingerprints and subject to human review, to limit hallucination [17]. It is not evaluated here.

IV. RESULTS

➢ *Performance*

Table 1 and Fig. 1 show the mean full-cycle time. At 4 KiB, OTP-OS is fastest (0.036 ms), followed by AES-GCM (0.057 ms) and ChaCha20-Poly1305 (0.060 ms). From 64 KiB, AES-GCM is fastest: 1.536 ms at 1 MiB and 6.901 ms at 5 MiB, against 10.999 ms for ChaCha20-Poly1305, 18.241 ms for OTP-OS and 22.062 ms for LCG-XOR at 5 MiB. GA-XOR needs 55.6 ms at 4 KiB and 120.4 ms at 5 MiB; a fixed search cost dominates, making it 1,554.9, 510.8, 45.7 and 17.4 times slower than the fastest method at the four sizes. Reporting encryption throughput alone would mislead: at 1 MiB GA-XOR encrypts at 657.5 MiB/s, but its round-trip rate is only 14.3 MiB/s.

Table 1 Mean Full-Cycle Time (ms), 30 Blocks Per Cell

Method	4 KiB	64 KiB	1 MiB	5 MiB
LCG-XOR	0.233	0.388	5.324	22.062
GA-XOR	55.641	56.671	70.145	120.376
OTP-OS	0.036	0.185	3.914	18.241
AES-GCM	0.057	0.111	1.536	6.901
ChaCha20-Poly1305	0.060	0.160	2.304	10.999
GA-XOR / fastest	1,554.9×	510.8×	45.7×	17.4×

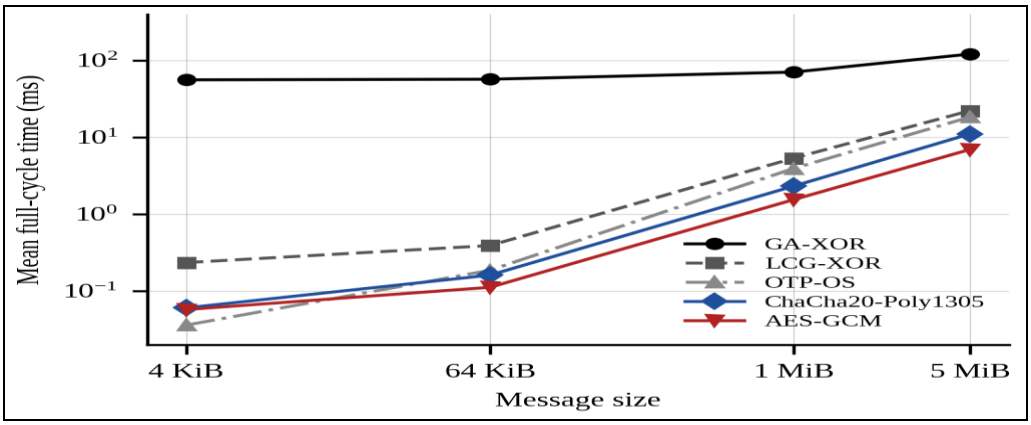


Fig 1 Mean Full-Cycle Time Versus Message Size (Log-Log Axes); GA-XOR Is Dominated by its Fixed Search Cost.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26oct012

➢ *Statistical Inference*

Friedman tests rejected equal ranks at every size (χ^2 = 110.9 to 120.0; Kendall's W = 0.924 to 1.000; Holm-adjusted $p < 10^{-22}$). AES-GCM and ChaCha20-Poly1305 showed no significant difference at 4 KiB (time ratio 0.938, 95% CI [0.832, 1.060], Holm p = 0.339), but AES-GCM was 30.8%, 33.4% and 37.2% faster at 64 KiB, 1 MiB and 5 MiB (ratios 0.692, 0.666 and 0.628; Holm $p \approx 2.2\times10^{-8}$). This ordering is specific to the tested platform and may change on processors without AES acceleration.

➢ *Integrity*

All 1,200 primary records were recovered correctly, so round-trip success cannot separate secure from insecure schemes. In the tamper test, both AEAD ciphers rejected 90 of 90 modified packages, whereas LCG-XOR, GA-XOR and OTP-OS returned silently altered plaintext in 90 of 90 cases (Table 2). Perfect secrecy of the OTP therefore does not imply integrity.

Table 2 Integrity and Keystream Diagnostics

Method	Tamper rejected	Silent changes	Secret size at 1 MiB (B)	Lag-256 match
LCG-XOR	0/90	90	8	1.0000
GA-XOR	0/90	90	32	0.0039
OTP-OS	0/90	90	1,048,576	0.0039
AES-GCM	90/90	0	32	n/a
ChaCha20-Poly1305	90/90	0	32	n/a

➢ *Keystream Diagnostics*

LCG-XOR attained the ideal entropy of 8.000000 bit/byte with $\chi^2 = 0$, yet its lag-256 correlation and match ratio were both 1.0: the keystream repeats every 256 bytes. GA-XOR (7.997075 bit/byte) and OTP-OS (7.997201 bit/byte) showed no such periodicity, with lag-256 matches near 1/256 ≈ 0.0039. High entropy is thus a necessary but insufficient indicator and does not establish unpredictability. Memory peaked at about 2,048 KiB for the AEAD ciphers and about 4,096 KiB for the XOR methods at 1 MiB; OTP-OS additionally requires a secret as long as the message, whereas AEAD needs a 32-byte key.

➢ *Added Value of the GA*

On the training sample, the GA reached a fitness of 0.99378 against 0.98672 for a single seed (Holm $p = 1.0\times10^{-5}$), but random search with the same 156-evaluation budget reached 0.99390 (GA versus random search: p = 0.259). On the independent hold-out sample the values were 0.98804 (GA), 0.98704 (random search) and 0.98632 (single seed), with no significant difference after correction (p = 0.259). Generation took 56.7 ms for the GA, 53.9 ms for random search and 0.005 ms for a single seed.

V. DISCUSSION AND LIMITATIONS

The results do not reject genetic algorithms in general. They show that this specific use, optimizing marginal statistics of a short sample of a SHAKE-256-derived stream, improves the metric it selects on without a demonstrated security gain, at 17 to 1,555 times the cost of the fastest alternative. Round-trip success, entropy and low correlation are descriptive indicators; none replaces a threat model, nonce management and cryptanalysis. Limitations: measurements come from one machine and a Python implementation; tracemalloc does not observe all native allocations; 90 tamper cases per method are not a proof of AEAD security and do not cover nonce misuse; OTP-OS is a baseline for key-size cost, not a key-distribution protocol; and the neural key-generation prototypes were not part of the quantitative comparison.

VI. CONCLUSION

We benchmarked OTP-OS, LCG-XOR, GA-XOR, AES-GCM and ChaCha20-Poly1305 under a randomized, statistically controlled protocol. Four findings emerge: the speed ranking depends on message size and on whether key-generation cost is included, with AES-GCM fastest from 64 KiB; only the AEAD ciphers detect tampering; entropy cannot expose a 256-byte keystream period; and the GA provides no demonstrated advantage over equal-budget random search. For protecting files, standard AEAD should be preferred over custom XOR keystreams. Future work should define GA objectives tied to a threat model, adopt multi-size hold-out protocols, repeat the benchmark on other hardware such as ARM, and measure resident memory with external tools.

ACKNOWLEDGMENT

This paper is derived from the first author's master's thesis at the Syrian Virtual University (Web Science Program), supervised by the second author.

REFERENCES

[1]. C. E. Shannon, "Communication theory of secrecy systems," Bell Syst. Tech. J., vol. 28, no. 4, pp. 656–715, 1949, doi: 10.1002/j.1538-7305.1949.tb00928.x.

[2]. C. Matt and U. Maurer, "The one-time pad revisited," in Proc. IEEE Int. Symp. Inf. Theory (ISIT), 2013, pp. 2706–2710.

[3]. M. Sokouti, B. Sokouti, S. Pashazadeh, M.-R. Feizi-Derakhshi and S. Haghipour, "Genetic-based random key generator (GRKG): A new method for generating more-random keys for one-time pad cryptosystem," Neural Comput. Appl., vol. 22, no. 7–8, pp. 1667–1675, 2013.

[4]. A. Jain and N. S. Chaudhari, "An improved genetic algorithm for developing deterministic OTP key generator," Complexity, vol. 2017, Art. no. 7436709, 2017, doi: 10.1155/2017/7436709.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26oct012

[5]. D. Apdilah, M. K. Harahap, N. Khairina, A. M. Husein and M. Harahap, "A comparison of one time pad random key generation using linear congruential generator and quadratic congruential generator," J. Phys.: Conf. Ser., vol. 1007, no. 1, pp. 1–6, 2018.
[6]. J. Jin and K. Kim, "3D CUBE algorithm for the key generation method: Applying deep neural network learning-based," IEEE Access, vol. 8, pp. 33689–33702, 2020, doi: 10.1109/ACCESS.2020.2973695.
[7]. Z. A. Ali and H. F. Hasan, "A review of artificial intelligence (AI) applications in key generation for encryption algorithms," Iraqi J. Sci. Eng. Res., vol. 4, no. 1, Art. no. 289, 2025, doi: 10.58564/IJSER.4.1.2025.289.
[8]. M. Bellare and C. Namprempre, "Authenticated encryption: Relations among notions and analysis of the generic composition paradigm," in Advances in Cryptology – ASIACRYPT 2000, LNCS vol. 1976. Berlin, Germany: Springer, 2000, pp. 531–545, doi: 10.1007/3-540-44448-3_41.
[9]. M. Dworkin, "Recommendation for block cipher modes of operation: Galois/Counter Mode (GCM) and GMAC," NIST, Gaithersburg, MD, USA, Spec. Publ. 800-38D, Nov. 2007, doi: 10.6028/NIST.SP.800-38D.
[10]. Y. Nir and A. Langley, "ChaCha20 and Poly1305 for IETF protocols," IETF, RFC 8439, Jun. 2018, doi: 10.17487/RFC8439.
[11]. L. E. Bassham et al., "A statistical test suite for random and pseudorandom number generators for cryptographic applications," NIST, Gaithersburg, MD, USA, Spec. Publ. 800-22 Rev. 1a, Apr. 2010.
[12]. M. Turčaník and M. Javurek, "Cryptographic key generation by genetic algorithms," Inf. Secur.: Int. J., vol. 43, no. 1, pp. 54–61, 2019, doi: 10.11610/isij.4305.
[13]. S. A. Shawkat, N. Tagougui and M. Kherallah, "Optimization-based pseudo random key generation for fast encryption scheme," Bull. Electr. Eng. Inform., vol. 12, no. 2, pp. 1007–1018, Apr. 2023, doi: 10.11591/eei.v12i2.4953.
[14]. G. R. Bansimba, R. F. Babindamana and P. A. G. Kidoudou, "A new approach in one-time pad key management," Int. J. Comput. Sci. Netw. Secur., vol. 19, no. 9, pp. 1–8, 2019.
[15]. D. E. Knuth, The Art of Computer Programming, Vol. 2: Seminumerical Algorithms, 3rd ed. Reading, MA, USA: Addison-Wesley, 1998.
[16]. H. W. Chung et al., "Scaling instruction-finetuned language models," J. Mach. Learn. Res., vol. 25, pp. 1–53, 2024.
[17]. Z. Ji et al., "Survey of hallucination in natural language generation," ACM Comput. Surv., vol. 55, no. 12, pp. 1–38, 2023, doi: 10.1145/3571730.