

Don't Escape The Matrix

A GPU-Parallelizable Matrix-Based Block Cipher with Two-Dimensional Diffusion

Prakhar S Mittal

University School Of Automation and Robotics
Guru Gobind Singh Indraprastha University, New Delhi

Abstract—We propose a matrix-based symmetric block cipher designed explicitly for parallel execution on graphics processing units (GPUs). Unlike the classical Hill cipher, which applies a single linear transformation $C = KP$, our construction applies a two-sided matrix transformation $ASB \bmod q$ within a substitution-permutation network (SPN), giving diffusion across both rows and columns of the state in a single step while remaining nonlinear overall. Because the cipher is operated in counter (CTR) mode, blocks are mutually independent and can be encrypted as a single batched matrix-multiplication call, mapping naturally onto GPU tensor/GEMM hardware. We present the formal construction, prove that the pure linear core is vulnerable to a known-plaintext attack (motivating the nonlinear layer), describe a GPU-oriented batched implementation, and outline an experimental methodology for correctness, statistical/cryptanalytic evaluation, and throughput benchmarking against AES-CTR on GPU.

Index Terms—block cipher, matrix cryptography, GPU cryptography, parallel encryption, Hill cipher, SPN

I. INTRODUCTION

Block ciphers such as AES operate on small, fixed-size blocks (128 bits) using round functions whose individual steps are inexpensive but whose parallelism is constrained by block size and mode of operation. Modern GPUs, in contrast, are optimized for large, regular linear-algebra workloads such as batched matrix multiplication. This mismatch motivates the question we address: can a block cipher be designed so that its core diffusion step is a matrix multiplication, and its mode of operation exposes block-level independence, so that encryption reduces to a batched GEMM (general matrix multiply) call that fully exploits GPU hardware?

The classical Hill cipher [1] already uses matrix multiplication as its encryption primitive, but it is purely linear and is broken trivially by a known-plaintext attack using $O(m)$ plaintext-ciphertext pairs [2]. Our contribution extends the Hill cipher's single-sided, one-dimensional matrix transform into a two-sided, two-dimensional diffusion step, embeds it inside a nonlinear SPN (in the spirit of AES/Rijndael), and evaluates its suitability for GPU execution specifically, rather than treating the GPU port as an afterthought as is common in prior GPU-AES work [3]–[6].

The contributions of this paper are:

- A formal, two-sided matrix diffusion step ($ASB \bmod q$) generalizing the Hill cipher to two dimensions, with a proof of why nonlinearity is required.

- A GPU-oriented key schedule producing round matrices with invertibility guaranteed by construction (as products of elementary matrices).
- A CTR-mode construction exposing full inter-block parallelism, implemented as a single batched matrix multiply on GPU.
- An experimental methodology and preliminary results for correctness, statistical/differential evaluation, and throughput versus CPU and AES-CTR-GPU baselines.

II. RELATED WORK

A. Matrix-based classical cryptography

The Hill cipher [1] encrypts an m -symbol block P as $C = KP \bmod q$ for an invertible key matrix K . It achieves full diffusion across the m symbols of a single block but is a linear transformation of the plaintext, so an attacker holding m linearly independent known plaintext-ciphertext pairs can recover K directly via Gaussian elimination [2]. This known weakness is the starting point for our design: we retain matrix multiplication as the diffusion primitive but wrap it in a nonlinear substitution layer, following the general substitution-permutation network (SPN) strategy used by Rijndael/AES [7].

B. GPU-accelerated block ciphers

A substantial body of work accelerates AES on GPUs via CUDA, generally by mapping the existing AES round structure onto GPU threads with varying granularity (bytes per thread) and memory layout choices. Di Biagio et al. [3] implemented AES-CTR on CUDA and reported a $14\times$ throughput improvement over CPU OpenSSL. Iwai et al. [4] studied the effect of memory allocation and parallelism granularity, achieving approximately 35 Gbps on an NVIDIA GTX 285. Li et al. [5] reported around 60 Gbps on a Tesla C2050 using shared-memory T-boxes. Abdelrahman et al. [6] studied multiple granularities across GPU generations, reporting up to 277 Gbps on a Pascal-generation GPU. In all of these works, AES's byte-oriented round structure (SubBytes, ShiftRows, MixColumns on 4×4 state) is mapped onto the GPU after the fact; none of them redesign the diffusion primitive itself to be a native GEMM operation. Our design instead makes the diffusion step itself a matrix multiplication from the outset, so that the mapping onto GPU batched-GEMM hardware (cuBLAS / tensor cores) is direct rather than an engineering accommodation.

III. PROPOSED ALGORITHM

A. State representation

A plaintext block is represented as a matrix $S \in \mathbb{Z}_q^{n \times n}$. We discuss the choice of modulus q in Section III-F.

B. Round function

Each round $k = 1, \dots, r$ applies, in order:

$$R_k(S) = \text{Permute}\left(\sigma\left(A_k(S \oplus K_k)B_k \bmod q\right)\right) \quad (1)$$

where K_k is the round key, σ is a fixed bijective S-box applied element-wise, $A_k, B_k \in \mathbb{Z}_q^{n \times n}$ are keyed invertible matrices, and Permute is a keyed row/column shuffle.

C. MixMatrix: two-dimensional diffusion

The core diffusion step is

$$S' = A S B \bmod q. \quad (2)$$

Left-multiplication by A mixes each *column* of S (a linear combination of rows); right-multiplication by B mixes each *row* of S (a linear combination of columns). Applying both in a single step means every output cell S'_{ij} is a function of every entry of S :

$$S'_{ij} = \sum_{p=1}^n \sum_{q'=1}^n A_{ip} S_{pq'} B_{q'j} \bmod q. \quad (3)$$

This is the two-dimensional generalization of the Hill cipher's single left-multiplication $C = KP$, which mixes only one axis.

D. Lemma: pure matrix mixing is linearly invertible

Lemma 1. *The map $S \mapsto ASB \bmod q$, with A, B fixed, is linear in S . Consequently, given a sufficient number of known plaintext-ciphertext matrix pairs with linearly independent structure, A and B (or their combined action) can be recovered by solving a linear system, analogous to the classical known-plaintext attack on the Hill cipher [2].*

Proof sketch. For fixed A, B , the map $S \mapsto ASB$ is linear because $A(S_1 + S_2)B = AS_1B + AS_2B$ and $A(cS)B = c(ASB)$ for scalar c . An attacker holding plaintext-ciphertext pairs can therefore set up a linear system in the unknown entries of A and B (or, if only recovering the equivalent round action is required, in the unknown entries of the combined linear operator) and solve via Gaussian elimination, exactly as in the one-sided Hill cipher case. ■

This lemma is the formal justification for including the nonlinear SubBytes layer (Section III-E): without it, Eq. 2 alone is cryptographically broken regardless of block size or choice of A, B .

E. SubBytes

Each entry is passed through a fixed bijective S-box $\sigma : \mathbb{Z}_q \rightarrow \mathbb{Z}_q$, applied element-wise:

$$S'_{ij} = \sigma(S_{ij}). \quad (4)$$

We reuse the AES S-box [7] rather than designing a new one. The AES S-box has well-studied nonlinearity and differential-uniformity properties; reusing it lets us cite existing analysis directly rather than re-deriving these properties for a novel S-box, and isolates our contribution to the matrix diffusion layer and the GPU-oriented construction.

F. Choice of modulus q

Two choices were considered:

- $q = 256$: byte-aligned, but $\mathbb{Z}_{256}$ has zero divisors, so only matrices with determinant coprime to 2 (odd determinant) are invertible.
- q prime (e.g. $q = 251$ or $q = 257$): $\mathbb{Z}_q$ is a field, so any matrix with nonzero determinant is invertible, which simplifies both the key schedule's invertibility guarantee and the correctness proofs.

For the reference implementation and initial experiments we use a prime modulus to keep invertibility reasoning simple; a byte-aligned mod-256 variant is left as an implementation optimization once correctness is established (see Section VIII).

G. Key schedule

The master key K_{master} and a per-message nonce are the only values supplied externally. All round material $\{K_k, A_k, B_k\}_{k=1}^r$ is derived deterministically:

$$\text{seed}_k = \text{HKDF}(K_{\text{master}}, k) \quad (5)$$

using a standard key-derivation function (HKDF [8] or SHA-256 in counter mode) rather than a novel PRF, so that the schedule's security properties can be argued from an established primitive rather than asserted.

Invertibility of A_k and B_k is guaranteed by *construction* rather than by rejection sampling: each matrix is built as a product of elementary matrices (row swap, row scaling by a unit, row addition of a multiple of another row) whose entries are derived from seed_k . Since a product of invertible matrices is always invertible, this avoids the need to compute a determinant and retry, and gives a one-line correctness argument for the paper.

H. Mode of operation: CTR

Rather than chaining blocks (as in CBC), each block's keystream is generated independently from the nonce and a counter:

$$S_i = P_i \oplus R_r(R_{r-1}(\dots R_1(\text{Nonce} \parallel i) \dots)) \quad (6)$$

Because block i 's computation does not depend on block $i - 1$'s output, all blocks in a message can be processed in parallel. This is the property that enables batched GPU execution (Section IV).

IV. GPU IMPLEMENTATION

A. Batched execution model

Because CTR mode removes inter-block dependencies, N blocks can be stacked into a single batched tensor of shape (N, n, n) and the MixMatrix step (Eq. 2) becomes a single batched matrix multiplication:

$$\text{Batch}' = A \cdot \text{Batch} \cdot B \quad (7)$$

executed via a library batched-GEMM call (e.g. cuBLAS `batched_gemm`, or the batched matrix multiply in PyTorch/CuPy), rather than N sequential calls. SubBytes becomes a single coalesced table-lookup (or, on GPU, an elementwise mapping) over the whole batch, and Permute becomes an index-remapping operation, both trivially parallel across the batch dimension.

B. Why block size matters for GPU occupancy

A larger block (e.g. $n = 16$, giving a 16×16 state versus AES's 4×4) increases the amount of independent arithmetic per block, which can improve GPU occupancy per batched-GEMM call, at the cost of a larger key schedule and larger per-block memory footprint. This trade-off is evaluated experimentally in Section VI.

V. EXPERIMENTAL METHODOLOGY

This section describes the experiments to be run (Section VIII provides companion scripts); Section VI reports results once available.

A. Correctness

For a range of block sizes n and round counts r , we verify $D_K(E_K(P)) = P$ for randomly sampled plaintexts and keys, confirming that the key schedule's invertibility guarantee holds in practice.

B. Statistical and cryptanalytic evaluation

- **Avalanche effect:** flip one input bit, measure the fraction of output bits that change (target $\approx 50\%$).
- **Key sensitivity:** fix the plaintext, flip one key bit, measure output bit-change fraction.
- **Differential trails:** for fixed input difference ΔP , measure the distribution of output differences ΔC across many keys/plaintexts, checking for non-uniform clustering.
- **Ciphertext randomness:** apply standard statistical randomness tests (e.g. frequency, runs, and serial tests from the NIST STS battery [9]) to ciphertext streams.

C. Performance benchmarking

Throughput (GB/s) and latency are measured as a function of batch size N and block size n , on both CPU (reference NumPy implementation) and GPU (batched PyTorch/CuPy implementation), and compared against AES-256-CTR (e.g. via a GPU AES library or OpenSSL as the CPU baseline), following the methodology of prior GPU-AES throughput studies [3], [4].

TABLE I

AVALANCHE AND DIFFERENTIAL TEST RESULTS, $n = 4$, 10 ROUNDS, AES S-BOX, $q = 257$ (200–500 TRIALS; SEE `CRYPTANALYSIS_TESTS.PY`)

Metric	Ideal	Measured
Plaintext avalanche (% bits changed)	$\approx 44.4\%$	45.0% ($\sigma=4.3\%$)
Key avalanche (% bits changed)	$\approx 44.4\%$	45.6% ($\sigma=4.1\%$)
Differential uniformity (max freq.)	low, uniform	0.20% (500/500 unique)
Monobit frequency (% ones)	$\approx 44.4\%$	44.0%
NIST STS pass rate	$\geq$ threshold	TODO (run full suite)

TABLE II

THROUGHPUT COMPARISON (FILL IN AFTER RUNNING `GPU_BENCHMARK.PY`)

Implementation	Throughput (GB/s)	Latency (ms / batch)
CPU reference (NumPy)	TODO	TODO
GPU batched (this work)	TODO	TODO
AES-256-CTR (GPU baseline)	TODO	TODO

VI. RESULTS

Results pending execution of the availability of High-End GPU clusters for computation of a throughput baseline across varied platforms; see Section VIII.

VII. DISCUSSION AND LIMITATIONS

This work proposes a construction and an evaluation methodology; it does **not** claim a formal security proof. Resistance to the tests in Section V is evidence against the specific attacks tested, not a guarantee against all attacks. In particular, side-channel behavior (timing and memory-access patterns) may differ materially on GPU versus CPU due to differing memory hierarchies, and is not evaluated in this version of the paper. We view formal security proof and side-channel analysis as important future work rather than omissions to be glossed over.

VIII. CONCLUSION AND FUTURE WORK

We presented a matrix-based block cipher whose diffusion primitive is a two-sided matrix multiplication and whose mode of operation exposes block-level independence, both chosen specifically to map onto GPU batched-GEMM execution. We proved that the linear core alone is vulnerable to known-plaintext attack, motivating the SPN construction, and described a GPU-oriented key schedule with invertibility guaranteed by construction. Future work includes: formal differential/linear cryptanalysis bounds for the specific S-box and matrix family chosen; side-channel evaluation on GPU; a byte-aligned mod-256 variant for direct hardware alignment; and extension of the optional Indic-script codebook layer discussed in preliminary design work, which remains a cosmetic layer rather than a security component.

REFERENCES

- [1] J. Overbey, W. Traves, and J. Wojdylo, "On the keyspace of the Hill cipher," *Cryptologia*, vol. 29, no. 1, pp. 59–72, 2005.
- [2] D. R. Stinson, *Cryptography: Theory and Practice*, 3rd ed. Chapman and Hall/CRC, 2005.

- [3] A. Di Biagio, A. Barengi, G. Agosta, and G. Pelosi, "Design of a parallel AES for graphics hardware using the CUDA framework," in *2009 IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*. IEEE, 2009, pp. 1–8.
- [4] K. Iwai, N. Nishikawa, and T. Kurokawa, "Acceleration of AES encryption on CUDA GPU," *International Journal of Networking and Computing*, vol. 2, no. 1, pp. 131–145, 2012.
- [5] Q. Li, C. Zhong, K. Zhao, X. Mei, and X. Chu, "Implementation and analysis of AES encryption on GPU," in *2012 IEEE 14th International Conference on High Performance Computing and Communication*. IEEE, 2012, pp. 843–848.
- [6] A. A. Abdelrahman, M. M. Fouad, R. Orjéšek, and H. M. Dahshan, "Analysis on the AES implementation with various granularities on different GPU architectures," *Advances in Electrical and Electronic Engineering*, vol. 15, no. 3, pp. 526–535, 2017.
- [7] National Institute of Standards and Technology, "Advanced encryption standard (AES)," FIPS PUB 197, 2001.
- [8] H. Krawczyk and P. Eronen, "HMAC-based extract-and-expand key derivation function (HKDF)," RFC 5869, 2010.
- [9] National Institute of Standards and Technology, "A statistical test suite for random and pseudorandom number generators for cryptographic applications," NIST Special Publication 800-22, 2010.