DisCrypt: An Ultra-Fast High-Throughput Encryption Engine for AES*

Nivedita Shrivastava
Department of Electrical Engineering
Indian Institute of Technology Delhi
New Delhi, India
Email: nivedita.shrivastava01@gmail.com

Smruti R. Sarangi
Department of Electrical Engineering
Indian Institute of Technology Delhi
New Delhi, India
Email: srsarangi@cse.iitd.ac.in

***Abstract*—AES is the most popular symmetric cryptographic algorithm as of today. There was no formal proof of its security guarantees until 2021. In the last five years, multiple papers have been published that prove the security of a generalized version of AES known as AES*, which is at least as secure as it. Moreover, the cryptographic community has proposed a novel metric known as *t-wise distance* that can be used to quantify security. AES* suffers from very large overheads – it requires 112% more area, which leads to a 82% reduction in throughput/area efficiency compared to an AES accelerator. Our contribution in this paper is to reduce its overheads and make a practically realizable version of the AES* algorithm. The crucial insight is to reduce the number of rounds to 2 by giving a smaller instantiation of the AES* algorithm a *head start*. Our implementation *DisCrypt* has a $5\times$ better latency and 70% higher throughput/area than an optimized AES accelerator. It is provably as secure as any AES-128 implementation.**

***Index Terms*—Hardware security, Secure Architecture, Encryption**

I. Introduction

The AES-NI [1] x86 instruction that provides hardware support to implement the AES algorithm takes 3.5 cycles per byte on most modern processors. This is quite slow for performance-sensitive applications [18]. To speed up the AES algorithm, it is necessary to quantify security as a function of the randomness of the input. Clearly, a random sequence of 0s as the input should require less encryption effort.

In the last few years, the AES* [6], [8], [9], [13] algorithm has been proposed that generalizes AES and is at least as secure as it. Its overheads are prohibitive; therefore, the aim in this paper is to create a practical implementation.

In this context, *t-wise independence* has been used as formal security metric [8]. It is defined as follows. Assume that the total variation (statistical) distance (TVD) between the joint distribution of t samples of the ciphertext and t randomly chosen samples from a theoretically uniform distribution is 0. In practice for a block of 128 bits, it is sufficient to show that the TVD $\leq \epsilon$ where ($\epsilon \leq 2^{-128}$) (ϵ-close). It is reducible to pairwise independence by appropriately modifying the distance between the plaintexts (not relevant for our design). For a window of t samples, it is equivalent to Shannon's perfect secrecy.

A. Problem solved in this paper: Practical AES*:

The main insight behind our work is that the effort required to encrypt plaintexts generated by a high-entropy random source should be much lower than the effort required to encrypt generic plaintexts. Given that we can now quantify security, and make it a function of the randomness of the input, this issue can be dealt with precisely.

The key insight is to give the AES* algorithm a head start. We optionally also use a PRNG, which is known to be weak (not strong enough for cryptographic purposes), to generate an input for a 2-round AES* network that has a feedback loop. The output is a mask, which is guaranteed to have a TVD $\leq 2^{-128}$ vis-a-vis the uniform distribution. This mask is encrypted again using a 2-round AES* network. The ciphertext protects the mask because its TVD with respect to it is $\leq 2^{-128}$, and can be used as a one-time pad (OTP), which is XORed with the plaintext. The critical path comprises only two cycles (encrypting the mask using a 2-round network). A speedup of $5\times$ is clearly visible here.

Security Guarantees

❶ AES* is at least as secure as conventional AES mainly because it uses distinct, uniformly random S-boxes for each byte in each round (proven in [8]).
❷ T-wise independence ensures that for any set of t plaintexts, the corresponding ciphertexts appear statistically independent and uniformly random (TVD $\leq \epsilon (= 2^{-128})$). Guaranteeing t-wise independence obviates the need for CCA, CPA, linear, non-linear and differential cryptanalysis [8].

The paper is organized as follows. §II provides the relevant background. §III presents a characterization of random number generators. §IV shows the proposed hardware design, §V contains the evaluation results and §VI discusses the related work. We finally conclude in §VII.

II. Background

A. Theory of Layouts

The distance between two probability distributions P and Q is typically quantified using the *total variation distance* (TVD).

$$\Delta_{TV} = \frac{\int_{x \in \mathcal{R}} |P(x) - Q(x)|}{2} \tag{1}$$

Our aim is to ensure that the 128-bit ciphertexts appear to be sampled from a uniform distribution, i.e., they carry no information about the underlying plaintext distribution. This is easily ensured by requiring the TVD between the joint distribution of t ciphertext samples to be ϵ-close to uniform, where $\epsilon \leq 2^{-128}$. If the ciphertexts are independent, then t-wise independence is easily relatable to pairwise independence. This is the case in our design, hence, we shall concern ourselves with pairwise independence subsequently. For the sake of readability, we shall assume that pairwise independence holds if the $TVD \leq 2^{-128}$ (*almost uniform* distribution or the random variable $\in TPI_\epsilon$). Similarly, a distribution (or random variable (r.v.)) is *sufficiently uniform* if its $TVD \leq 2^{-50}$ (w.r.t. uniform).

The *layout* $\mathbf{L}(\mathbf{y_1}, \mathbf{y_2})$ computed between two k-element vectors $\mathbf{y_1}$ and $\mathbf{y_2}$ is defined as follows:

$$\forall i \in [1,k] : \mathbf{L}[i] = \begin{cases} 1 & \mathbf{y_1}[i] = \mathbf{y_2}[i] \\ 0 & \mathbf{y_1}[i] \neq \mathbf{y_2}[i] \end{cases} \tag{2}$$

The most important result in a series of papers [2], [4], [9] is the following:

$$\Delta TV\left((\mathbf{y_1}, \mathbf{y_2}), (\mathbf{u_1}, \mathbf{u_2})\right) = \Delta_{TV}\left(\mathbf{L}(\mathbf{y_1}, \mathbf{y_2}), \mathbf{L}(\mathbf{u_1}, \mathbf{u_2})\right) \tag{3}$$

The following relationship is self-evident: $(\mathbf{y_1}, \mathbf{y_2}) \in TPI_\epsilon \Leftrightarrow (\mathbf{L}(\mathbf{y_1}, \mathbf{y_2}), \mathbf{L}(\mathbf{u_1}, \mathbf{u_2})) \in TPI_\epsilon$. We can thus reason in terms of layouts, whose dimension is limited to $2^{16} - 1$ ($\because k = 16$ and $\mathbf{y_1} \neq \mathbf{y_2}$). Our aim is to reduce ϵ to less than or equal to 2^{-128}, which means that the information obtained from one round of related plaintext/ciphertext analysis is less than what one would obtain by assigning the key a random value.

B. AES*

Consider classical AES-128 that represents. 16 bytes (128 bits) using a 4×4 matrix. The SubBytes operation substitutes each byte with another byte (to achieve non-linearity) using a *common* S-box. The next operation is ShiftRows: shift the i^{th} row by i positions to the right. Next, we have MixColumns: multiply each column by a known matrix $\mathcal{M}$. Finally, the AddRoundKey operation computes a XOR of the current state of the 4×4 matrix with a 16-byte round key. The first round has an additional AddRoundKey operation and the last round omits the MixColumns operation. It is important to note that that the process of generating round-specific keys (key schedule) from an original key is not considered to be a cryptographically strong operation [8]. because it is a sequence of algebraic operations.

AES* thus omits the key schedule and the AddRoundKey operations. Instead, it uses different and independent S-boxes for every byte in every round. S-box contents are truly independent across bytes and rounds in the case of AES*. We however need 160 S-boxes here because these structures are not shared.

Observation: AES* uses more random and independent information as compared to classical AES; this enhances its security at the cost of significantly higher memory overheads..

In AES*, each round can be represented as a transformation function $\mathcal{F}$ that changes the layout. The final aim is to prove that the layout distribution matches the distribution of layouts that will be produced by a uniform distribution of ciphertexts. This translates to a binomial distribution $\mathcal{B}$ on layouts. It has also been proven in [9] that the variation distance after a round is a function of $|\mathbf{L}|$ (number of 1s in the input layout). Note that we will sometimes abuse terminology a little, and represent $\mathbf{L}(\mathbf{x}, \mathbf{y})$ as $\mathbf{L}$, when the inputs are irrelevant. Higher the value of $|\mathbf{L}|$, the more similar are the plaintexts, and the higher is ϵ (after one round). To realize $\epsilon \leq 2^{-128}$, we need more rounds. A sequence of rounds is like a Markov chain where the space of layouts is the state space.

The random variable here is a layout. After r rounds (Markov transitions), we shall have a final distribution of layouts. The stationary distribution should ideally be a binomial distribution of layouts. Our aim is to find the number of rounds r that ensures that the final variation distance vis-a-vis $\mathcal{B}$ is $\leq 2^{-128}$. We follow the optimized method of Liu et al. [9] to compute the TVD.

III. Characterization

A. Characterization of the Total Variation Distance (TVD)

In this section, we shall rigorously analyze the TVD of the final ciphertext layouts vis-a-vis the stationary binomial distribution for inputs with different degrees of randomness, and track it across rounds. The input to the Markov chain is a distribution of layouts. We can characterize any layout distribution by its TVD with respect to the binomial distribution. As long as the TVD is the same, the specific distribution of layouts did not appear to have any statistically significant effect. The results for different starting TVDs is shown in Figure 1.

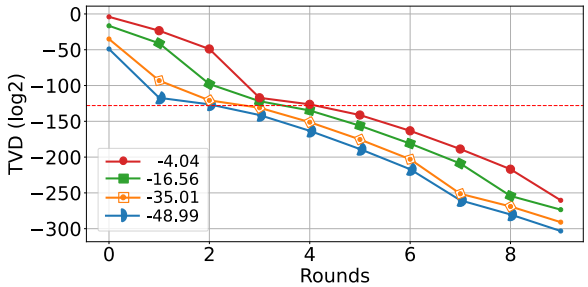

Fig. 1: Variation of the TVD of the ciphertext layout distributions with rounds. Different colored lines represent different initial TVD values of the input layout distributions.

As per expectations, we observe that TVD consistently decreases as the number of encryption rounds increases. For

input distributions with an initial TVD less than 2^{-50} (sufficiently uniform), we find that **two rounds of encryption** are sufficient to reduce the TVD to $\leq 2^{-128}$.

Insight

If we can somehow ensure that the input is sufficiently uniform, then we need only 2 rounds of encryption to make the output almost uniform.

If we compute the XOR of a sample drawn out of an almost uniform distribution with some plaintext T, then the output is also almost uniform as per the XOR lemma [5]. This is true regardless of the randomness in T. Hence, the problem gets reduced to generating an almost uniform mask that can be XORed with plaintexts. However, a known plaintext attack is possible, and the mask can get leaked. Hence, we need to generate a new almost uniform mask for every 128-bit plaintext, which is quite challenging.

Problems

- P1 Create a unique and independent mask for every encryption. The mask needs to be almost uniform.
- P2 Ensure that Step (1) is performed very quickly.
- P3 Protect the mask from being exposed even if P1 is solved. This is because we don't want the attacker to get access to the mask and derive useful information by analyzing the sequence of masks.

B. Sketch of the Solution

The crux of our idea is as follows. We observe in Figure 1 that encryption in AES* requires 7-10 rounds for arbitrary plaintexts; however, it will only take 2 rounds if the plaintext is sufficiently uniform ($\epsilon \leq 2^{-50}$). Let us start with an initialization vector (IV), which is unknown. Then, let us repeatedly encrypt it. Given that the input is almost uniform, a short 2-round encryption using AES* will continue to produce almost uniform outputs. This can be proven using mathematical induction. This mechanism can be further bolstered by XORing the output of a good PRNG (random number generator) with the mask. Even though the PRNG is not strong enough, by itself, for cryptographic purposes; nevertheless, it can enhance the randomness of each mask that is produced. This will make it practically very difficult to recover the contents of the S-boxes even if the IV or mask are exposed. Given that we require 2 encryption rounds, we have a solution for problems P1 and P2. P3 is still not solved.

C. Characterization of the Random Number Generators

The objective of this section is to characterize the performance of different kinds of PRNGs: chaos-based and classical designs. We generate over 10^9 random numbers and measure the performance using a machine with the following configuration: Intel 2.9 GHz Xeon 6326 CPU, 8 GB DRAM, 64 cores, Ubuntu 20.04. Note that the performance is computed as the reciprocal of the execution time.

We study 6 different PRNGs – *XoroShiro* [16]; *Permutation Congruential Generators(PCG)* [12]; *Linear Congruential Generators(LCG)* [7]; *Mersenne Twister* [10](MT); *Stnd. C++* in-built PRNGs. We also analyze the performance of 5 state-of-the-art chaotic maps as listed in Table I. Chaotic maps are mathematical functions that exhibit sensitive dependence on initial conditions, leading to unpredictable and complex behavior over time. The performance results of PRNGs and chaotic maps are shown in Figure 2.

Scheme	**Description**
XoroShiro [16]	Combines OR, shifts and rotations.
LCG [7]	Generates numbers using a linear recurrence relation.
PCG [12]	Uses LCG with output transformation.
MT [10]	Recursively transforms states using bitwise operations.
std::rand()	Standard C++ random number generator.
std::mt19937	Standard C++ implementation of MT.
Chaotic Map	**Equations**
Henon [14]	$x_{i+1} = 1 + y_i - ax_i^2$; $y_{i+1} = bx_i$
Tent [14]	$x_{i+1} = \mu x_i$ if $x_i < 1/2$; $x_{i+1} = \mu(1-x_i)$ if $x_i > 1/2$
Logistic [14]	$x_{i+1} = \mu x_i(1 - x_i)$
Lorenz [14]	$x_{i+1} = \sigma(y_i - x_i)$; $y_{i+1} = x_i(\rho - z_i) - y_i$; $z_{i+1} = x_i y_i - \beta z_i$
Chirikov [14]	$x_{i+1} = x_i + ksin(y_i)$; $y_{i+1} = y_i + x_{i+1}$

TABLE I: RNG techniques. (x, y, z) represents random map values while other variables represent control parameters.

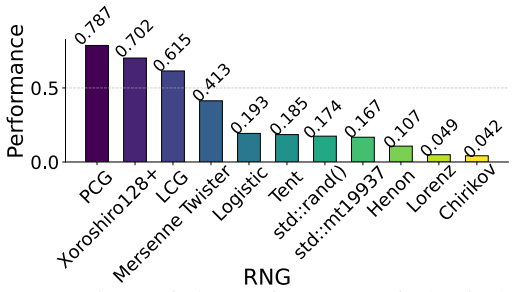

Fig. 2: Comparison of the performance of classical pseudo random number generators and chaotic maps

Among the conventional PRNGs, *PCG* and the *XoroShiro* are the fastest. While, for chaotic maps, the *Logistic* map is the most computationally efficient, while the *Chirikov* and *Lorenz* maps exhibit the slowest performance. The degraded performance of the *Chirikov* map is due to the *sine* computation. The *Lorenz* map also performs complex computations, which degrade its performance.

Conclusion

The *PCG* and *XoroShiro* algorithms exhibit high performance and are lightweight. Chaos-based RNGs were not found to be suitable due to their poor performance.

IV. Design of DisCrypt

DisCrypt comprises two stages (refer to Figure 3). Stage A contains the mask generation circuit (problems P1 and P2). Stage B solves problem P3 by further encrypting the mask. It also requires 2 rounds because the mask is uniformly random. The output is a one-time pad (OTP) that can be XORed with the plaintext.

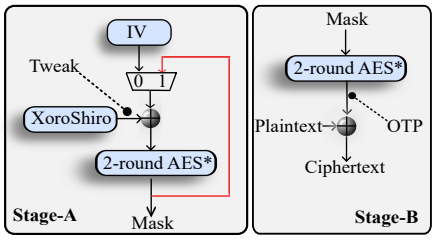

Fig. 3: Overall design of *DisCrypt*

A. Stage A: Mask Generation Engine

1) Mask Generation: Figure 3 shows a feedback circuit for mask generation (as discussed in Section III-B). We additionally XOR the output of *XoroShiro128+* to the current mask. It plays the role of a salt. The *XoroShiro128+* block has two 8-byte internal states, $s0$ and $s1$, which are initialized to the seed of the random number generator. XoroShiro uses the XOR, OR and shift operations as shown in Figure 4. The output in each clock cycle is the sum of $s0$ and $s1$. We run two parallel and independent *XoroShiro* engines with different seeds to generate two 8-byte random numbers concurrently, and then concatenate them to generate a 16-byte random *tweak*. The engine operates with a period of 2^{128}, which ensures that the PRNG can produce 2^{128} distinct values before repeating, thereby significantly reducing the likelihood of collisions.

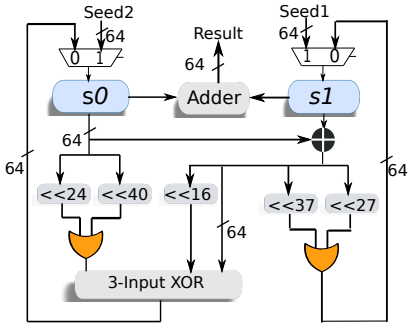

Fig. 4: *XoroShiro* engine for generating the *tweak*

To generate a new mask for every plaintext encryption instance (problem P1), we add a feedback loop. It takes into account the additional salt generated by the PRNG. This salt is XORed with the previous mask and the result is encrypted using AES* to generate the next mask, further ensuring that the generated output is always *almost uniform*. This mask generation process is executed before the actual encryption process (Stage B) begins.

B. Stage B: Encryption Engine

In Stage B, the mask generated in Stage A is encrypted again using a 2-round AES* engine to generate a 128-bit mask, which functions as the OTP. The advantage of this stage is that it solves Problem P3 and protects the mask. This OTP is then XORed with the plaintext to produce the final ciphertext.

While Stage B is actively encrypting the current plaintext using the already available mask, Stage A simultaneously generates the mask for the next encryption. This parallel processing ensures that the mask generation does not add any latency to the encryption process, as the system is always ready with a new mask when needed.

C. AES* Encryption Engine

AES* has a 128-bit data path, and requires only two rounds of encryption. Thus, for each round, we need 16 parallel and independent S-boxes to encrypt each data byte. We use the same *MixColumn* and *ShiftRow* operations as AES for permuting the data.

There are different approaches for the generation of such random secret S-boxes; we can use any one of them. ① Use random numbers generated from the RNG as the S-boxes. The risk here is that PRNGs are a weak source of random numbers. ② Derive S-boxes using a cryptographic algorithm such as AES. ③ Use hard-coded S-boxes. This is not recommended. ④ Let the user specify the contents. It is assumed that the user would have used a strong source of random numbers to generate them.

V. Evaluation

A. Setup and Configurations

We evaluate the two most optimized versions of classical AES as the baselines: *pipelined* AES with unrolling factors of 3 (AES-3x) and 4 (AES-4x), where the hardware for computing a single round is replicated three and four times, respectively. We also evaluate AES-XTS (tweakable block cipher, used in Intel SGX), the baseline AES* design with 160 S-boxes for 10 encryption rounds (AES*-10r), and the popular lightweight cipher PRESENT. Note that PRESENT is not as strong as AES variants, and is thus not suitable for most applications; we have included it to serve as an academic point of comparison. Note that the block size of all these configurations except PRESENT is 128-bit. We use two parallel 64-bits PRESENT engines to generate a 128-bit output. This was done to ensure that the input and output sizes of all the designs are the same.

The code for all configurations was written in Verilog and the designs were synthesized, placed, and routed using the Cadence Genus tool (28-nm ASIC technology node).

B. Results

The overall results for each configuration are shown in Table II. We observe that *DisCrypt* is the most efficient with the least latency and the highest throughput. *DisCrypt* is nearly $5\times$, $9\times$ and $4\times$ faster than AES, AES-XTS and PRESENT, respectively. *DisCrypt* offers a 70.58% improvement in throughput/area compared to pipelined AES. It is $5.37\times$ more efficient than AES-XTS in terms of throughput/area. This is because we have decreased the number of cycles and eliminated latency-critical operations from the critical path. AES-XTS (used by Intel SGX [3]) has the maximum latency since it requires two sequential AES engines to encrypt a block of plaintext.

PRESENT achieves a higher throughput-to-area ratio, making it $3.2\times$ more efficient than *DisCrypt*. It is a lightweight cipher and has its share of vulnerabilities (quite unlike AES and

Module	Area (μm^2)	Thr. ($Gbps$)	Latency (ns)	Energy (pJ)	Thr/area ($Mbps/\mu m^2$)
AES-3x	13575.49	11.52	27.77	20.96	0.85
AES-4x	18075.77	15.36	27.77	21.20	0.85
AES*-10r	28865.59	4.35	29.41	122.35	0.15
AES-XTS	9321.00	2.52	50.76	65.48	0.27
PRESENT	1175.20	5.46	23.43	3.53	4.64
DisCrypt	**15405.39**	**23.04**	**5.65**	**6.68**	**1.45**
Tool	Cadence RTL Compiler, 28 nm				

TABLE II: Comparative analysis of the encryption schemes

AES*). Also, *DisCrypt* achieves nearly $10\times$ higher throughput/area, $5\times$ lower latency and 47% better area compared to AES*-10r.

C. Discussion on Security

Note that no modifications were made to the AES* algorithm. The security of AES* is derived from extensive cryptanalytic studies conducted over the last two decades, along with rigorous theoretical proofs that establish the *t-wise* independence of AES* [8], [9]. The code for computing the required number of rounds is based on computing the TVD after every round (derived from the Python code provided by Liu et al. [9]).

It is easy to see using mathematical induction that if the original input is almost uniform (TVD $\leq 2^{-128}$) (refer to Figure 1), then in every stage, this invariant is being maintained. At no point of time is the invariant (TVD $\leq 2^{-128}$) being violated because the input is always almost uniform. The typical range of computed TVDs was between 2^{-128} and 2^{-192}.

D. Use Cases and Limitations

The only case in which we shall not get good results is when the key changes for every encryption. In that case, our bootstrapping overheads are high, and we need to encrypt at least 675 bytes to break even. However, we are not aware of any such practical scenario where this happens. In all cases, it takes time to establish a key and subsequently that key is used for a reasonably long time (eg: TLS or SSL based encryption).

VI. Related Work

Ueno [15] introduces a highly efficient on-the-fly key schedule and a novel architecture for AES-CBC, which employs operation-reordering and register-retiming techniques to unify the two stages of encryption - SubBytes and InvSubBytes. Similarly, Mozaffari et al. [11] also propose a low-latency and high-throughout parallel and pipelined design for the AES-GCM mode. Unlike us, they don't fundamentally speed up AES.

Yang et al. [17] highlight the inadequacy of traditional S-boxes for image encryption due to the high correlation present in image data. To mitigate this issue, they propose a new S-Box generation algorithm based on a 4-D hyperchaotic system with an enhanced particle swarm optimization method. Zhang et al. [19] also propose using quantum random walks governed by a hyper-chaotic map for constructing dynamic S-Boxes. These approaches based on chaotic maps are not cryptographically robust unlike our approach.

VII. Conclusion

We achieved our objective of creating a fast, high-throughput accelerator. We outperform AES by $5\times$ in the terms of latency, while offering 70% improvement in throughput/area. Given that *DisCrypt* is simply an implementation of AES*, it did not need a separate security proof; we could simply leverage well-known results that provide t-wise independence. Our novelty relied on creating an architecture that provides a head start, reduces the length of the critical path, uses a time-varying mask that changes on every encryption, and effectively hides it (along with the masked+encrypted PRNG output) using two additional layers of encryption.

VIII. Acknowledgements

We would like to thank the Ministry of Electronics and IT (India) for supporting us with a grant to carry out this research.

References

[1] "Securing the enterprise with intel aes-ni," Intel Corporation, Tech. Rep., 2010, accessed: 2023-09-23. [Online]. Available: https://www.intel.com/content/dam/doc/white-paper/enterprise-security-aes-ni-white-paper.pdf
[2] T. Baignoires, "Proving the security of aes substitution-permutation network," *Available at: http://lasecwww. epfl. ch*.
[3] V. Costan and S. Devadas, "Intel sgx explained." *IACR Cryptol. ePrint Arch.*, vol. 2016, no. 86, pp. 1–118, 2016.
[4] J. Daemen, "The design of rijndael: Aes-the advanced encryption standard," *Springer google schola*, vol. 2, pp. 432–445, 2002.
[5] O. Goldreich, "Three xor-lemmas—an exposition," *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation*, pp. 248–272, 2011.
[6] R. Gradwohl and A. Yehudayoff, "t-wise independence with local dependencies," *Information processing letters*, vol. 106, no. 5, 2008.
[7] D. E. Knuth, *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*, 3rd ed. Boston, MA, USA: Addison-Wesley, 1997.
[8] T. Liu *et al.*, "The t-wise independence of substitution-permutation networks," in *Advances in Cryptology–CRYPTO*. Springer, 2021.
[9] ——, "Layout graphs, random walks and the t-wise independence of spn block ciphers," in *Annual Int. Cryptology Conf.* Springer, 2023.
[10] M. Matsumoto and T. Nishimura, "Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator," *TOMACS*, vol. 8, no. 1, pp. 3–30, 1998.
[11] M. Mozaffari-Kermani and A. Reyhani-Masoleh, "Efficient and high-performance parallel hardware architectures for the aes-gcm," *IEEE TC*, vol. 61, no. 8, pp. 1165–1178, 2012.
[12] M. E. O'neill, "Pcg: A family of simple fast space-efficient statistically good algorithms for random number generation," *ACM TOMS*, 2014.
[13] A. Pelecanos, "Non-asymptotic t-wise independence of substitution-permutation networks," Ph.D. dissertation, MIT, 2022.
[14] D. Singh *et al.*, "A systematic literature review on chaotic maps-based image security techniques," *Comput. Sci. Rev.*, vol. 54, p. 100659, 2024.
[15] R. Ueno *et al.*, "A high throughput/gate aes hardware architecture by compressing encryption and decryption datapaths," in *CHES*, B. Gierlichs and A. Y. Poschmann, Eds. Springer, 2016, pp. 538–558.
[16] F. Vigna, "Xorshift random number generators," 2021, accessed: 2024-09-22. [Online]. Available: https://vigna.di.unimi.it/xorshift/
[17] S. Yang *et al.*, "S-box generation algorithm based on hyperchaotic system and its application in image encryption," *Multimedia Tools and Applications*, vol. 82, no. 17, pp. 25 559–25 583, 2023.
[18] T.-H. Yoo *et al.*, "Avx-based acceleration of aria block cipher algorithm," *IEEE Access*, vol. 11, 2023.
[19] L. Zhang *et al.*, "A novel dynamic s-box generation scheme based on quantum random walks controlled by a hyper-chaotic map," *Mathematics*, vol. 12, no. 1, p. 84, 2023.