IACR Communications in Cryptology
ISSN 3006-5496, Vol. 1, No. 2, 44 pages.
https://doi.org/10.62056/a3n59qgxq

Twinkle: A family of Low-latency Schemes for Authenticated Encryption and Pointer Authentication

Jianhua Wang[1], Tao Huang[2], Shuang Wu[2] and Zilong Liu[3]

[1] Shield Lab, Huawei Technologies Co., Ltd., China
[2] Shield Lab, Huawei International Pte. Ltd., Singapore
[3] HiSilicon Technologies Co. Ltd., China

Abstract. In this paper, we aim to explore the design of low-latency authenticated encryption schemes particularly for memory encryption, with a focus on the temporal uniqueness property. To achieve this, we present the low-latency Pseudo-Random Function (PRF) called `Twinkle` with an output up to 1152 bits. Leveraging only one block of `Twinkle`, we developed `Twinkle-AE`, a specialized authenticated encryption scheme with six variants covering different cache line sizes and security requirements. We also propose `Twinkle-PA`, a pointer authentication algorithm, which takes a 64-bit pointer and 64-bit context as input and outputs a tag of 1 to 32 bits.
We conducted thorough security evaluations of both the PRFs and these schemes, examining their robustness against various common attacks. The results of our cryptanalysis indicate that these designs successfully achieve their targeted security objectives.
Hardware implementations using the FreePDK45nm library show that `Twinkle-AE` achieves an encryption and authentication latency of 3.83 ns for a cache line. In comparison, `AES`-CTR with WC-MAC scheme and Ascon-128a achieve latencies of 9.78 ns and 27.30 ns, respectively. Moreover, `Twinkle-AE` is also most area-effective for the 1024-bit cache line. For the pointer authentication scheme `Twinkle-PA`, the latency is 2.04 ns, while `QARMA-64-`σ_0 has a latency of 5.57 ns.

Keywords: Low-latency · Authenticated Encryption · Pointer Authentication

1 Introduction

The landscape of symmetric-key cryptography has witnessed a notable evolution in recent years, moving from general-purpose designs to domain-specific solutions tailored to meet the unique demands of specific applications. The National Institute of Standards and Technology (NIST) has been a pioneer in this evolving landscape, with its efforts to standardize lightweight cryptography, aiming to provide cryptographic primitives optimized for resource-constrained scenarios. An earlier instance of this transition is evident in the previous CAESAR competition [com], where the final portfolio encompassed three distinct use cases: lightweight applications, high-performance applications, and defense in depth, each requiring a nuanced cryptographic design approach.

One domain-specific aspect that has received significant attention in recent years is low-latency cryptography, particularly in the areas of memory protection and system security enhancement.

E-mail: wangjianhua@amss.ac.cn (Jianhua Wang), huangtao80@huawei.com (Tao Huang), Wu.Shuang@huawei.com (Shuang Wu), liuzilong5@hisilicon.com (Zilong Liu)

Received: 2024-04-09 Accepted: 2024-06-03

Memory protection. In the context of cloud computing, memory is a vulnerable target, susceptible to physical and privileged software attacks, including threats posed by cloud providers [HSH+09, BPH15, YADA17, WCJ+21]. To address these vulnerabilities, it is essential for Central Processing Units (CPUs) to incorporate cryptographic mechanisms to safeguard the data stored in memory. A natural approach involves the implementation of a hardware memory encryption engine, strategically positioned between the system cache and Random Access Memory (RAM). The cipher is intended to be implemented as the Memory Encryption Engine (MEE) on the SoC, situated between the DRAM Controller and the Caches, as shown in Figure 1. When data is loaded from DRAM, the MEE will decrypt and verify its integrity. Conversely, when data is written to the DRAM, the MEE will perform encryption and generate a tag, which may be stored in the DRAM (either in the memory itself or by repurposing the ECC bits in DRAM).

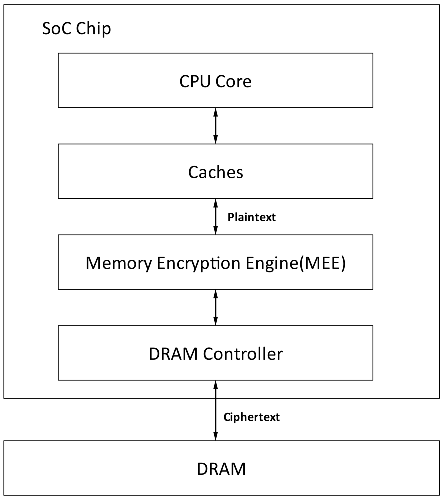

Figure 1: Schematic diagram of memory encryption engine.

Various memory protection solutions have been proposed, such as Intel Software Guard Extensions (SGX) [Gue16], Intel Trust Domain Extension (TDX) [Int20], AMD SEV[AMD19], and ARM Confidential Compute Architecture (CCA) [ARM21]. These solutions differ in their security properties, and Avanzi [Ava22] has classified them into three levels, from basic memory encryption to memory encryption, authentication and replay protection.

- **Level 1: Memory encryption.** This level provides memory confidentiality using a tweakable block cipher with the address as tweak. AMD's SEV is an example of Level 1 solutions.

- **Level 2: Encryption and integrity verification.** This level adds integrity protection against memory corruption to Level 1. It does not protect against replay attacks. Intel's TDX is an example of Level 2 solutions.

- **Level 3: Encryption, integrity and replay protection.** This level provides stronger protection against replay attacks. Note that while the nonce based encryption can provide temporal uniqueness, it is still vulnerable to replay attacks. To prevent replay attacks, normally an integrity tree is needed and the root of the integrity tree is stored inside the SoC. Intel's SGX is an example of Level 3 solutions.

The temporal uniqueness of ciphertext is a critical factor in mitigating side-channel attacks, as emphasized in previous research [LWW+22, DLT+23]. These studies have revealed vulnerabilities in Trusted Execution Environments (TEEs) that use deterministic memory encryption, highlighting the necessity of nonce-based authenticated encryption schemes to address these attacks. Due to the importance of Level 3, this paper will introduce an authenticated encryption primitive designed for this level. Besides, throughout this paper, we assume a strong attacker can perform active attacks with physical access to DRAM, allowing them to read and modify data. Cryptographically speaking, the adversary can modify the ciphertext and tags.

System security enhancement. Cryptographic primitives are now being used to enhance system security, with Pointer Authentication (PA) being one such primitive. PA effectively prevents software attacks that rely on modifying pointers, such as Return-Oriented-Programming (ROP) attacks, Data-Oriented-Programming (DOP) attacks, Out Of Bound read/write (OOB) and Used-After-Free (UAF) attacks. Additionally, low-latency small block ciphers have the potential to be useful in performing cache randomization, which helps prevent side-channel attacks.

There are two different approaches to protect the integrity of a pointer. One approach is to use a MAC algorithm to compute a tag for the pointer and its context. This approach is particularly useful for 64-bit pointers that have unused bits which can be used to store the MAC tag. It provides a deterministic probability to detect attacks on integrity. ARM has adopted this idea and provides the pointer authentication code (PAC) after ARMv8.3.

Another approach is to use encryption with redundant information in the plaintext such as the pointer address. Since the addresses of valid pointers are only a subset of the entire space, when a pointer is encrypted with a block cipher, the probability of a modified ciphertext decrypting to a valid address is undeterministic. However, it should be smaller than using the redundant bits as MAC tag, as it captures those invalid addresses with correct redundant values. In the Cryptographic Capability Computing (C^3) [LRD+21] proposed by LeMay et al., a 24-bit tweakable block cipher is needed to encrypt a segment of a pointer.

The design of low-latency cryptographic primitives is a relatively new research field. One of the earliest low-latency block ciphers is `PRINCE` [BCG+12], which was published in 2012 and optimized for hardware latency. In 2020, an updated version called `PRINCEv2` [BEK+20] was proposed with improved security. `MANTIS` [BJK+16] is another low-latency tweakable block cipher with a 64-bit block size that combines the TWEAKEY framework and low-latency properties. `QARMA` [Ava17] is a family of low-latency tweakable block ciphers inspired by `PRINCE` and `MANTIS`, offering both 64-bit and 128-bit block sizes. A recent version called `QARMAv2` [ABD+23] was introduced, which supports larger tweaks and includes a version for pointer authentication and memory integrity. `K-Cipher` [KDGD20] is a proposed cipher that supports a range of block sizes from 24-bit to 1024-bit, but recent cryptanalysis work by Mahzoun et al. [MKPA22] suggests that its security margin may not be sufficient. `SPEEDY` [LMMR21] is a family of low-latency block ciphers that prioritize latency optimization at the cost of area and energy. The design uses a newly proposed low-latency 6-bit S-box and 192-bit block size. `Orthros` [BIL+21] is a low-latency PRF that can be used as building blocks for low-latency schemes. Followed the design strategy of `Orthros` which utilizes branches of permutation to form a PRF, another low-latency PRF `Gleeok` [ABC+24] was proposed recently to support a 256-bit key size. `SCARF` [CGL+23] is a low-latency block cipher designed specifically for cache randomization, using a 240-bit key size and 10-bit block size to achieve extremely low latency. Recently, Inoue et al. proposed a new memory encryption scheme, ELM[IMO+22], which includes a low latency MAC and Authenticated Encryption (AE) based on AES/AES round function. For pointer authentication, a new low-latency tweakable block cipher called `BipBip` [BDD+23] has been proposed, which is suitable for use in Intel's C^3. BipBip incorporates novel ideas such as a non-linear tweakey schedule, heterogeneous rounds, and a large masterkey size.

Despite the numerous low-latency schemes proposed in recent years, there still exists a gap between academic research and industrial solutions for memory protection. One of the issues contributing to this gap is the lack of standardization for low-latency ciphers. Industrial products typically require standardized cryptographic schemes, which means that even if a low-latency scheme performs better, it cannot be adopted in real products. This issue was prominently addressed by researchers from Intel and Google during the recent NIST workshop on lightweight cryptography[Gho22, Yal22]. They highlighted the critical need for standardized low-latency cryptography to bridge the gap between theoretical research and industrial application. We believe that the pressing demand for these applications will serve as a catalyst for establishing future low-latency standards within standard organizations such as NIST or ISO, ensuring broader adoption and technological integration.

Another issue is that existing low-latency designs primarily focus on encryption rather than authentication. As previously discussed, for memory protection of Level 2 or higher, it is important to implement both encryption and authentication mechanisms. Currently, most solutions combine encryption with authentication schemes, such as Intel SGX using AES-CTR for encryption and WC-MAC for authentication, and Intel TDX using AES-XTS for encryption and SHA-3 MAC for authentication. While replacing AES in these schemes with a low-latency block cipher is an option, careful design of the authentication algorithm remains critical. For instance, a narrow-block cipher might fall short of fulfilling the security criteria for the WC-MAC within the SGX scheme, and the latency associated with SHA-3 in the TDX scheme may not align with performance expectations. Besides, given the unique demands of memory encryption scenarios, the adoption of a dedicated authenticated encryption algorithm is anticipated to markedly reduce latency.

1.1 Our Contributions

In this research, we aim to address the question of how to design a low-latency authenticated encryption scheme specifically for memory encryption with temporal uniqueness. This feature is essential for mitigating ciphertext side-channel attacks and attaining Level 3 protection, as outlined by Avanzi [Ava22].

Departing from the ELM scheme by Inoue et al.[IMO+22], which leverages AES as its foundation to establish a tweakable block cipher, our approach is innovative. We have developed a novel AE scheme, `Twinkle-AE`, from the ground up. This includes the introduction of a new PRF with an expanded state, which we have adapted into a nonce-based AE scheme. In addition, we have proposed `Twinkle-PA`, a new low-latency MAC designed for pointer integrity, utilizing the PRF.

Furthermore, we recognize the significance of the recently introduced authenticated encryption schemes based on `Gleeok`, notable for their low-latency characteristics. `Gleeok` stands out by supporting a 256-bit key length and exhibiting flexibility for efficiently processing short message inputs. Our design primarily focuses on cache line encryption with a relatively fixed input size. When encrypting 1024-bit cache lines, our message to state size ratio approximates 0.9, in contrast to `Gleeok`'s 0.33. This distinction renders our design a more area-efficient option for the specified application.

We summarize the main contributions in our designs of `Twinkle-AE` as follows:

- **A novel AE construction with single PRF call.** At a high level, `Twinkle-AE` can be considered as a stream cipher with a Wegman Carter MAC (WC-MAC). The innovative concept involves using a single low-latency PRF to produce both the keystream for encryption and the random mask for authentication. As a result, the encryption and authentication only require the latency of one XOR operation in addition to the latency of the PRF. Despite its apparent simplicity, this idea incorporates several key insights specific to this scenario.

- **Time-aligned plaintext encryption/ciphertext decryption.** In the memory encryption scenario, the plaintext length is deterministic, which is the cache line size. With this in mind, we process all plaintext/ciphertext simultaneously for low latency goal.
- **Parallelizable authentication algorithms.** We utilized the WC-MAC, which combines a nonce-based random mask with a universal hash function (UHF). This choice offers low latency and parallelizability compared to integrated solutions like ASCON.
- **Plaintext/ciphertext-free computation.** The nonce-based design enables us to create encryption schemes as stream ciphers, which generate the keystream without relying on plaintext/ciphertext. In WC-MAC, the mask generation dominates the delay of MAC generation and is also plaintext-independent. These designs bring two advantages in latency.
  * When the nonce is available in advance, the keystream and mask can be pre-calculated without waiting for the plaintext/ciphertext to be loaded, resulting in the fastest authenticated encryption/verified decryption.
  * During the decryption verification process, the mask can also be calculated along with the keystream before the plaintext is restored. This minimizes the delay gap between decryption and encryption to only a low-delay UHF level, almost 0 if the nonce is always available in advance.
- **Single PRF for both encryption and authentication.** We have created a low-latency wide-size PRF that can generate both the keystream for encryption and random mask for authentication in a single call. This approach aligns the generation latency of keystream and mask, while maintaining the same level of security for both. Besides, using a single PRF simplifies designs and analysis compared to using multiple PRFs.

With all the above considerations, we believe that this design structure is close to being optimal, if it is not already the best.

- **A new PRF optimized for latency.** We summarize our efforts to optimize the design of PRF as follows:
  - **Tailored state size.** The state size of `Twinkle-AE` is 1280 bits. This choice offers several benefits. First, it can generate up to 1152-bit keystream for encryption and random mask for authentication by a single PRF call. This facilitates processing the entire cache line through one PRF call, streamlining both the design and cryptanalysis processes. Secondly, the relatively large state size enhances the differential/linear properties by allowing a greater number of active S-boxes in each round, thereby improving security. Additionally, with at least 128 bits concealed, the PRF's resilience against differential and linear analysis, impossible differential attacks, and guess-and-determine attacks is significantly enhanced. Lastly, the ratio of the output size to the state size is up to 0.9, closely optimizing area effectiveness.
  - **Enhanced diffusion of input data.** To expedite the mixing of input data, we employ diffusion functions to process the input data, allowing the input data to initially influence more bits of the internal state. We use n (where $n > 1$) permutation matrices with minimal latency overhead in hardware implementation to form the diffusion function. This operation also enhances

resistance against differential attacks, increasing the differential branch number to $n + 1$ from 2.

- **Hardware-efficient Round function.** We have developed a hardware-efficient round function, denoted as $\mathcal{R}$, that operates on the state of a cube structure. Our design optimizations focus on enhancing security while minimizing forward delay. We have chosen an asymmetric-latency diffusion matrix to boost resistance against differential and linear attacks. To further enhance this resistance, we have paired this matrix with a relatively low-latency S-box which has good cryptographic properties. Additionally, we have selected double bit-level lane rotation operations with minimal latency to accelerate diffusion and bolster security.

- **A new low-latency MAC scheme.** The `Twinkle-PA` is designed using the PRF `Twinkle` with the context and pointer as input. The `Twinkle` permutation can handle large amounts of data simultaneously with very efficient diffusion, resulting in the low-latency feature of the `Twinkle-PA`. Moreover, when implemented on a chip that already has the circuit of `Twinkle-AE`, the additional area needed for `Twinkle-PA` is very small.

`Twinkle-AE` family has six variants, which are listed in Table 1. `Twinkle-AE-512` family supports memory encryption with a 512-bit cache line, and the versions of which provide 128-bit, 128-bit, and 256-bit confidentiality security and 64-bit, 128-bit, and 128-bit authentication security, respectively. `Twinkle-AE-1024` family has versions with the same level of security as the `Twinkle-AE-512` family, but is designed for CPUs with a cache line size of 1024. `Twinkle-PA` reuses part of structure of `Twinkle-AE` and has inherited some good security properties. Furthermore, there will be little overhead for `Twinkle-PA` if the `Twinkle-AE` has been equipped in the chip.

Table 1: Security claim of `Twinkle-AE` versions

Versions		Confidentiality (bits)	Integrity (bits)
`Twinkle-AE-512`	`Twinkle-AE-512a`	128	64
	`Twinkle-AE-512b`	128	128
	`Twinkle-AE-512c`	256	128
`Twinkle-AE-1024`	`Twinkle-AE-1024a`	128	64
	`Twinkle-AE-1024b`	128	128
	`Twinkle-AE-1024c`	256	128

As a result, both `Twinkle-AE-512a` and `Twinkle-AE-1024a` achieve the low-latency goal, whose delay are only about **39.1%** of that of the authentication encryption scheme in Intel's SGX scheme (`AES` and WC-MAC) for the same security. And the latency of `Twinkle-AE-512b` and `Twinkle-AE-1024b` is only about **14.0%** of that of lightweight authentication encryption algorithm Ascon-128a.

Besides, the latency of `Twinkle-PA` is about only at most **36.6%** of that of `QARMA-64` family which is used in the ARMv8.3-A ISA extensions for pointer authentication.

1.2 Organization

We have organized our paper as follows. First, in Section 2, we introduce preliminaries and notations used in this paper. Section 3 specifies the low-latency PRF `Twinkle`. Then, we apply the `Twinkle` PRF to memory encryption and pointer authentication scenario in Section 4. Next we discuss the design rationales in Section 5. In Section 6, we state the security claim and provide the security evaluation for all versions of `Twinkle`, `Twinkle-AE` and `Twinkle-PA`. We present the results of hardware implementation for `Twinkle` and corresponding authentication encryption schemes and compare them with other low-latency ciphers in Section 7. Finally, we conclude our paper in Section 8.

2 Preliminaries

2.1 Operations

The following operations are used in this paper.
$\oplus$: bitwise exclusive OR.
&: bitwise AND.
$\sim$: bitwise NOT.
$\|$: concatenation. For Example, $110||01 = 11001$
$\lll$: rotation to the left. For Example, $(01110011 \lll 2) = 11001101$.
$\ggg$: rotation to the right. For Example, $(01110011 \ggg 2) = 11011100$.
$n \pmod m$: For integers n and m, $n \pmod m$ is the integer $r \in \{0, \cdots, m-1\}$ so that $n - r$ is divided by m.

2.2 Notations

For a natural number m, $\mathbb{N}_{<m}$ denotes the set $\{0, 1, \cdots, m-1\}$.

If the length of a bit string S is m, then its bits are indexed from 0 to $m-1$, i.e. $S = S[m-1]||\cdots||S[1]||S[0]$. The least significant bit of the bit string S is $S[0]$. The bit string S of length $16 \times l$ could be described as a $4 \times 4 \times l$ three-dimensional array. The expression $S[x][y][z]$ with x, $y \in \mathbb{N}_{<4}$, and $z \in \mathbb{N}_{<l}$, denotes the bit in position (x, y, z) of state S. Besides, the expression $S[x][y][z]$ in the three-dimensional array is equivalent to the expression $S[x + 4y + 16z]$ in the one-dimensional array for the bit string S. The expression $S[\bullet][y][z]$, $S[x][\bullet][z]$ and $S[x][y][\bullet]$, respectively, denotes the row indexed by (y, z), the column indexed by (x, z) and the lane indexed by (x, y) of state S. For details, see Figure 2. The expression $S[\bullet][\bullet][z]$, $S[\bullet][y][\bullet]$ and $S[x][\bullet][\bullet]$, respectively, denotes the slice indexed by (z), the plane indexed by (y) and the sheet indexed by (x) of state S. The bit string S of length $16 \times l$ also could be denoted as $S_d^{d-1}||\cdots||S_d^1||S_d^0$, where d divides l and S_i^d of length $16 \times l/d$ is the i-th substring of S, i.e. $S_d^i[j] = S[16il/d + j]$ for $i \in \{0, \cdots, d-1\}$ and $j \in \{0, \cdots, 16l/d - 1\}$. Moreover, S_d^i could be reshaped to a $4 \times 4 \times l/d$ three-dimensional array like S.

For a permutation σ of $\mathbb{N}_{<m}$, $\mathbf{P}_\sigma$ is represented as a permutation matrix corresponding to σ so that $\mathbf{P}_\sigma[i, j] = 1$ if and only if $i = \sigma(j)$, otherwise $\mathbf{P}_\sigma[i, j] = 0$, where $\mathbf{P}_\sigma[i, j]$ is the element in the i-th row and j-th column of $\mathbf{P}_\sigma$. For the sake of simplicity of notations, the bit string S of length m also could be viewed as the vector in $\mathbb{F}_2^m$, i.e. $S = [S[0], S[1], \cdots, S[m-1]]^T$. Then $\mathbf{P} \cdot S = \mathbf{P}[S[0], S[1], \cdots, S[m-1]]^T$. The result of $\mathbf{P} \cdot S$ which is a vector in $\mathbb{F}_2^m$ also could be viewed as a bit string $(\mathbf{P}S)[m-1]||\cdots||(\mathbf{P}S)[0]$.

2.3 Wegman-Carter MACs

Wegman-Carter (WC) type MACs [WC81, Ber05, CS16] use a *UHF-then-PRF* design, the message is hashed by a universal hash function (UHF), and then the hash value is encrypted by a nonce-based mask $f_{k_2}(N)$ to generate a tag. The formula is:

$$T = h_{k_1}(M) + f_{k_2}(N), \tag{1}$$

where k_1 is the key for the UHF h, k_2 is the key for the PRF f, M is the message, and N is the Nonce.

The security of WC-MACs is guaranteed only when nonce is respected. Assuming that f_{k_2} is a perfect uniformly random function, the adversary honestly queries the oracle at most

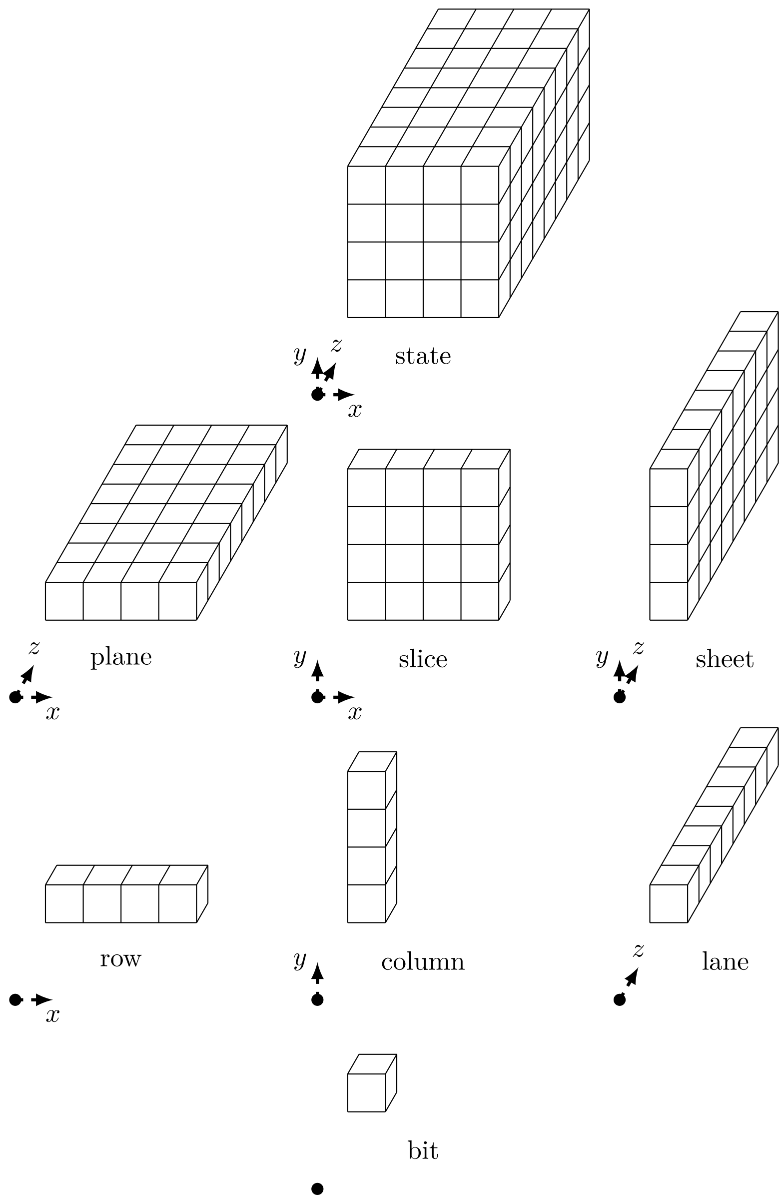

Figure 2: Schematic diagram of each component of 128-bit 3D array

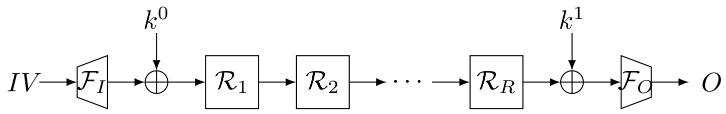

Figure 3: The overview of `Twinkle`

q times with messages and distinct nonces, and the oracle will return the corresponding tags. In this case, the success probability of the adversary's at most p forgery attempts is at most $p\varepsilon$ [CS16], where ε is the maximal differential probability of h_{k_1}, namely,

$$\varepsilon = \max_{M \neq M', X} \{h_{k_1}(M) + h_{k_1}(M') = X\}.$$

If f is not perfect, the adversary will gain the advantage in distinguishing f from a uniformly random function within $p + q$ queries.

3 Specification of PRFs

This section introduces `Twinkle`, a wide-size low-latency PRF, the overview of which is illustrated in Figure 3. The structure of `Twinkle` is an Even-Mansour scheme in addition to an input expansion operation $\mathcal{F}_I$ and an output compression operation $\mathcal{F}_O$. The central permutation is formed by the round function $\mathcal{R}$, and the number of rounds is determined by security requirements.

The sizes of internal states for `Twinkle` is 1280, denoted by ρ. The state S is a three-dimensional array of elements of $\mathbb{F}_2$, with dimensions $4 \times 4 \times l$, where $4 \times 4 \times l = \rho$.

3.1 Key Scheduling Function

The whitening keys k^0 and k^1 are both derived from the same key K, and have a length equal to the internal state size ρ. k^0 is set to K, while k^1 is calculated as $(K \ggg 1) \oplus (K \gg \rho - 1)$.

3.2 Round Function

The round function $\mathcal{R}$ is designed using the SPN structure. The permutation $\mathcal{R}$ is a sequence of operations performed on the state S, specifically,

$$\mathcal{R} = \mathbf{AC} \circ \mathbf{LaneRotation}_1 \circ \mathbf{MixSlice} \circ \mathbf{LaneRotation}_0 \circ \mathbf{S\text{-}box}.$$

Each component updates the state S as follows:
S-box: A 4-bit S-box **Sb** is applied to every row of the state S in parallel. Namely,

$$S[0][y][z]||\cdots||S[3][y][z] \leftarrow \mathbf{Sb}(S[0][y][z]||\cdots||S[3][y][z]),$$

where $y \in \mathbb{N}_{<4}$ and $z \in \mathbb{N}_{<l}$. The specification in hexadecimal is shown in the following table.

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$\mathbf{Sb}(x)$	0	3	5	d	6	f	a	8	b	4	e	2	9	c	7	1

LaneRotation$_0$: A rotation operator is applied to every lane of the state S in parallel. Namely,

$$S[x][y][\bullet] \leftarrow S[x][y][\bullet] \ggg (O_0[x + 4y] \mod l)\,,$$

where $x, y \in \mathbb{N}_{<4}$.

MixSlice: A linear diffusion operation is applied to every slice of the state S in parallel. Namely,

$$S[\bullet][\bullet][z] \leftarrow S[\bullet][\bullet][z] \oplus (S[\bullet][\bullet][z] \lll 5) \oplus (S[\bullet][\bullet][z] \lll 12),$$

where $z \in \mathbb{N}_{<l}$.

LaneRotation$_1$: Another rotation operator is applied to every lane of the state S in parallel. Namely,

$$S[x][y][\bullet] \leftarrow S[x][y][\bullet] \ggg (O_1[x+4y] \mod l)\,,$$

where $x, y \in \mathbb{N}_{<4}$.

Table 2: The offsets of **LaneRotation**

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
O_0	20	24	38	77	49	66	30	40	76	15	46	50	17	18	61	62
O_1	63	45	34	39	32	43	60	66	54	26	55	36	61	12	15	35

AC: The state will be XORed with the round constant. The specification of the i-th round constant RC_i can be found in Appendix B.

3.3 Input Expansion Operation

The expansion operation takes a 128-bit input and expands it to 1280 bits using 10 bit-permutations denoted by σ_i (where $i = 0, \cdots, 9$). Specifically, the expansion operation $\mathcal{F}_I$ maps IV to S as follows:

$$S \leftarrow P_{\sigma_9} \cdot IV || \cdots || P_{\sigma_0} \cdot IV, \tag{2}$$

where $P_{\sigma_0} \cdot IV$ is padded to the least significant 128 bits of S, and so on. Each bit-permutation maps the set of natural numbers less than 128 ($\mathbb{N}_{<128}$) as follows:

$$j \mapsto (aj + b) \pmod{128}, \text{ for } j \in \mathbb{N}_{<128}.$$

The parameters a and b for each σ are listed in Table 3.

Table 3: The parameters of σ_i

i	0	1	2	3	4	5	6	7	8	9
a	1	3	5	7	11	13	17	19	23	29
b	0	1	2	3	4	5	6	7	8	9

3.4 Output Compression Operation

The output length of `Twinkle` is variable, varying from 1 to $\rho - 128$. The compression operation $\mathcal{F}_O$ is represented as

$$O \leftarrow \mathrm{Trun}_n\left(S \oplus (S \gg 128)\right), \tag{3}$$

where Trun_n represents the least significant n bits of a bit string.

3.5 The Number of Rounds

According to various security goals, we have established different rounds as outlined in Table 4. The 0.5-round $\mathcal{R}$, specified by **LaneRotation$_0$ $\circ$ S-box**, is equipped at the end. Note that there is an extra condition for achieving 64-bit security, which limits the

Table 4: The number of rounds corresponding to different security levels

Security Level	256 bits	128 bits	64 bits†
# Rounds	18.5	9.5	5

† The output length is limited to 64 bits.

output length to at most 64 bits. In Section 6, we will explain how these variants offer corresponding security level within these parameters.

We use the notation $\texttt{Twinkle}_n^m$ to distinguish different variants based on their security and output size, where m represents the security level in bits and n represents the output size in bits.

4 Application of `Twinkle`

In this section, we apply `Twinkle` to memory encryption and pointer authentication solution, i.e., `Twinkle-AE` and `Twinkle-PA`.

4.1 Specification of the `Twinkle-AE`

For L3 level protection, a unique nonce is required for time freshness. This means that a block cipher is not necessary, and a PRF such as our `Twinkle` can be used. Our `Twinkle` can generate output up to 1152 bits wide, which is enough for the cache line sizes of Mainstream CPUs (512 or 1024 bits). It can serve as a keystream generator to produce a keystream, which can then be XORed with plaintext to produce ciphertext. To prevent the adversary from forging the ciphertext, the authentication process is necessary, and we have chosen the Carter-Wegman MAC as the authentication algorithm.

The `Twinkle-AE` family consists of six versions, as listed in Table 1, designed for two different cache line sizes and offering varying levels of security for both the confidentiality and integrity of plaintext. For simplicity, let c denote the cache line size. Let m be the confidentiality security level in bits. Let t represent the tag size, which is also equivalent to the integrity security level in bits in this case.

4.1.1 Encryption and Authentication

We instantiate WC-MAC with PRF as `Twinkle` and UHF as finite field multiplication in F_2^t. Since $c+t$ is less than 1152, it is sufficient to use one `Twinkle` function to output $c+t$ bits, within c bits for encryption and t bits for authentication. Specifically, the cipher C and the tag T is generated as follows:

$$\begin{aligned} O_t||O_c &= \texttt{Twinkle}_{c+t}^m(IV, K) \\ H &= \sum_{i=0}^{c/d-1} M_i \otimes K'_i \\ C &= O_c \oplus M \\ T &= O_t \oplus H \end{aligned} \tag{4}$$

where $M = M_{c/t-1}||\cdots||M_0$ is the message, K is 1280-bit master key for `Twinkle`, $K' = K'_{c/t-1}||\cdots||K'_0$ is another c-bit key, using for UHF in WC-MAC, O_c is the least significant c-bit of the `Twinkle`'s output, $\otimes$ represents the multiplication in F_2^d, and the length of tag, t is equal to d in this case.

4.1.2 Decryption and Verification

In the decryption process, first compute the message M by XORing the ciphertext C with the keystream O_c.

$$O_t||O_c = \texttt{Twinkle}^m_{c+t}(IV, K)$$
$$M = O_c \oplus C$$

After that the tag T' could be computed using the new message M.

$$H = \sum_{i=0}^{c/t-1} M_i \otimes K'_i$$
$$T' = O_t \oplus H$$

If the tag T' is equal to T, the verification will succeed and the message M will be output. Otherwise, the verification will fail and the newly generated message M and authentication tag T' should not be output.

4.2 Specification of `Twinkle-PA`

`Twinkle-PA` is a pointer authentication algorithm that takes a 64-bit pointer PT, a 64-bit context CT, and a 1280-bit secret key K as inputs to produce an authentication tag ranging from 1 to 32 bits. The tag T is generated using $\texttt{Twinkle}^{64}_t$ as follows:

$$T = \texttt{Twinkle}^{64}_t(CT||PT, K).$$

`Twinkle-PA` aims to provide 64-bit security against offline attacks and t-bit security against online attacks, where t is the length of the tag.

5 Design Rationale

The `Twinkle-AE` family is designed for Level 3 scenarios, requiring memory confidentiality, integrity, and temporal uniqueness as described in Section 1. `Twinkle-PA` is a pointer authentication algorithm for system security enhancement. This section will explain the design strategy used by the `Twinkle` family to meet these requirements, including decisions regarding the overall structure and individual components.

5.1 Construction of `Twinkle-AE`

In memory encryption scenarios, the length of the message is the same as the cache line size and does not exceed 1024 bits, which is different from typical authentication encryption scenarios. When using an integrated encryption and authentication structure like Keccak [BDPA13] and Ascon [DEMS21], the proportion of the delay in generating the authentication code to the overall delay will be very significant. (When the message is longer, this ratio becomes almost negligible.) Therefore, when designing the `Twinkle-AE` structure, we opted for a parallel encryption and authentication scheme instead of the integrated structure.

5.1.1 The Way of Encryption and Authentication

Encryption. To ensure temporal uniqueness in encryption, adopting a nonce-based approach is essential, which positions stream ciphers as a good choice. Stream ciphers, capable of generating a keystream independent of the plaintext or ciphertext, are ideal for

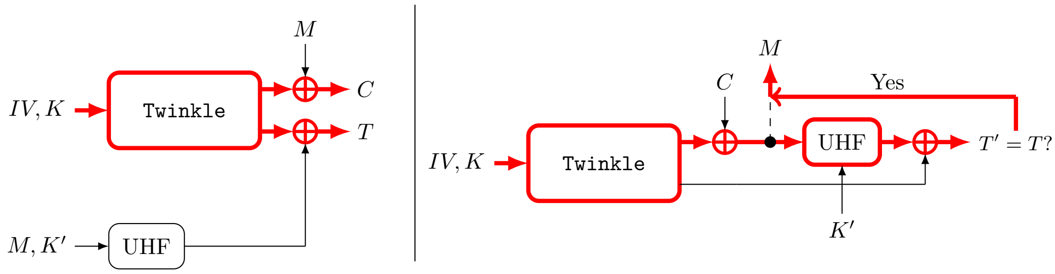

Figure 4: The critical path in encryption and authentication process(left) and decryption and verification process(right)

pre-computing keystream to accelerate the encryption and decryption processes. These ciphers can be constructed using either PRPs or PRFs. Only the forward functions will be utilized for both encryption and decryption. This approach simplifies delay considerations by eliminating the need to consider for delays in the inverse operations of S-boxes or matrices within PRPs/PRFs, thereby offering greater flexibility in the selection of S-boxes and matrices.

In comparison to PRPs, PRFs' irreversible nature provides additional analytical benefits, such as resistance to chosen ciphertext attacks and the increased complexity of meet-in-the-middle and linear attacks. Intuitively, PRFs are capable of achieving comparable levels of security with lower delays than PRPs, making them a preferred option for encryption design.

As plaintext/ciphertext length does not exceed 1024 bits, designing a PRF whose output covers the requirement of entire keystream allows for parallel processing of all plaintext or ciphertext, optimizing delay.

Authentication. The Wegman-Carter MAC (WC-MAC) [WC81] offers better parallelism and lower latency compared to standard hash functions, making it a strong choice. The WC-MAC involves XOR operations on the hash value of plaintext and a Nonce-based random mask, with mask generation being a key factor in latency. Pre-computing the mask when the nonce is known in advance can speed up authentication process.

We use a linear combination of finite field multiplication as the UHF and have developed a wide-output PRF to support the WC-MAC mask as well as the keystream for encryption. This design offers strong security beyond the birthday bound [BKR98, HWKS98] with minimal overhead compared to block cipher constructions for PRFs. It maintains low latency and simplifies security analysis by relying on the same PRF proof.

Critical Path. The critical path in the encryption and authentication process, as well as the decryption and verification process, is illustrated in Figure 4. Benefiting from parallel designs, the delay in the authentication and encryption (AE) process is determined by `Twinkle`'s delay plus an XOR's delay. On the other hand, the verification and decryption (VD) process has the delay a UHF's delay plus an XOR's delay longer than that of the AE process. Thus, the UHF's delay is also crucial for the VD process, which is also another reason for choosing WC-MAC. As for the critical path for the case where the nonce is available in advance, please refer to the Appendix C.

5.1.2 The Long Key

In our design, a secret key (K and K') with large length (equal to the state size 1280 plus the cache line size) is used, which is not a common choice. In most of the symmetric-key designs, the key size is minimized in terms of the security level (i.e. the same as the security level). This brings the advantage on the cost of key generation, key transmission and key storage over the ciphers with large key sizes (e.g. RSA).

In our targeted application scenario, the situations are different. There is no key transmission. For key storage, it requires a few thousand more gates to store the key, which is not a significant amount for a mainstream CPU. For key generation, it can be more efficient to generate large keys directly with the built-in hardware RNGs (such as RDRAND and RDSEED for Intel CPU, and RNDR and RNDRRS for ARM CPU) than using a complicated key expansion function. Meanwhile, it saves the circuits to implement the key expansion function.

We also noted the following benefits for a large key used in our design:

- Reduced computation for initialization phase.
- Simplified analysis. Doing key expansion during the initialization phase would lead to more consideration of the dependency.

5.1.3 The Tag Length.

We believe that the 64-bit authentication security is adequate for the majority of our targeted applications. In the case of forgery attacks, the adversary's ability to successfully forge is not increased by collecting the tuple of plaintexts, ciphertexts, and tags (assuming the number of collections is insufficient to recover the key). As a result, the adversary can only rely on a "blind guessing" strategy with a success probability of 2^{-64}, and the expected time for a successful forgery is approximately 2240 years (calculated as $2^{64} \times 3.83$ ns) for the computation time of `Twinkle-AE`, without considering the time for machine interrupts and restarts. This time frame is considered sufficient for most applications. For comparison, Intel SGX uses 56-bit tags for authentication.

Additionally, we offer 128-bit authentication security variants for specific use cases that require a higher level of security on the authentication.

5.2 Construction of Twinkle-PA

There are two main approaches to protecting pointer integrity: encryption and MAC. Encryption does not require extra storage space for the tag and can use redundant information to authenticate pointer addresses. However, the MAC method can be more efficient due to its large state size and one-way functions.

In this paper, we have chosen the MAC approach for its high effectiveness. We have developed Twinkle-PA by incorporating 5 rounds of Twinkle, which offers robust security and low latency. If the design of Twinkle-AE is already embedded in the chip, there is minimal overhead to embed the circuit of Twinkle-PA, only requiring the circuit for XORing whitening keys and tag generation.

5.3 Design of `Twinkle` Function

The `Twinkle` function uses a wide-size Even-Mansour construction as its main structure, employs diffusion mapping $\mathcal{F}_I$ to absorb 128 bits of input, and utilizes a compression function $\mathcal{F}_O$ to extract variable-length output ranging from 1 to 1152 bits.

5.3.1 Reason for 1-round Even-Mansour Construction

The Even-Mansour structure is a block cipher scheme proposed by Even and Mansour [EM93, EM97]. The ciphertext C is computed as follows:

$$C = k^1 \oplus P(M \oplus k^0), \tag{5}$$

where P is a PRP. For a multi-round Even-Mansour structure like AES, a key scheduling algorithm is required to generate a round key for each round. In our case, the round key length is 1280 bits, which results in a significant cost in chip area for implementing the key scheduling algorithm. In the next section, we will demonstrate that the provable security of single round of Even-Mansour structures have met our security requirements.

5.3.2 State size

The state size of `Twinkle-AE` is set to 1280 bits for several reasons. The output must accommodate both the keystream and random mask, totaling 1152 bits. To bolster resistance against attacks like differential analysis, linear analysis, impossible differential attacks and guess-and-determine attacks, additional non-output redundancy is necessary. However, the redundancy size must be balanced to maintain area efficiency. Therefore, the state size is chosen as 1280 bits.

5.3.3 Processing Input Data

For the large internal state of `Twinkle`, it is crucial for the input IV to diffuse quickly. To achieve this, we employ a technique of pre-diffusing low-dimensional input data into high-dimensional space.

While replicating multiple copies appear to be the most straightforward and low-overhead method, our analysis shows that it can lead to a correlation between the output, potentially compromising the security of the algorithm. This issue is explained in Example 1.

To address this, we introduce bit permutations to disrupt the order of the copies. This operation has negligible latency in hardware evaluation. Therefore, we propose the expansion operation $\mathcal{F}_I$ as shown in Equation 2. In this case, each bit of input data will influence 10 bits of the internal state. This enhances resistance to differential attacks, with the first 2-round differential path including at least 43 active S-boxes, compared to only 4 without diffusion. This little-latency diffusion has significantly improved security.

Example 1. Assuming that v is an element in $\mathbb{F}_2^{128}$, let's denote two copies of v as w, i.e. $w = v||v$. Consider the operation at the beginning of the `Twinkle`, which involves XORing and S-box operation. Each S-box operation $\mathbf{S}$ acts on the nibble with indices ranging from $4j$ to $4j+3$, where $j \in \mathbb{N}_{j<64}$. Specifically, we have $s_{4j}, \cdots, s_{4j+3} = \mathbf{S}(w_{4j} \oplus k_{4j}, \cdots, w_{4j+3} \oplus k_{4j+3})$.

Note that the adversary does not know the value of the fixed k, but he can control the value of v. Since the difference between the input of the j-th S-box and the $(j+32)$-th S-box is always equal to $\Delta_j = (k_{4j} \oplus k_{4j+128}, \cdots, k_{4j+3} \oplus k_{4j+3+128})$, the difference of outputs always lies in the set $\{S(x) \oplus S(x \oplus \Delta_j) : x \in \mathbb{F}_2^4\}$.

5.3.4 Output Generation

The `Twinkle` has a maximum output of 1152 bits and does not support the full-state 1280-bit output. This is because the authentication encryption scheme only requires a specific length, not exceeding 1152 bits. Besides, the remaining secret 128 bits can enhance resistance to attacks such as meet-in-the-middle attacks, differential attacks, and linear attacks and guess and determine attacks.

5.4 Design of $\mathcal{R}$

The security and latency of the $\mathcal{R}$ have a direct impact on the overall performance of the `Twinkle`. In designing the $\mathcal{R}$, we prioritize both low latency and meeting the security boundary principle by minimizing the number of rounds. Since the Feistel structure only updates a portion of the state in each round, we prefer to use the fully updated SPN structure in the $\mathcal{R}$.

Recall that the function $\mathcal{R}$ is defined as:

$$\mathcal{R} = \mathbf{AC} \circ \mathbf{LaneRotation}_1 \circ \mathbf{MixSlice} \circ \mathbf{LaneRotation}_0 \circ \mathbf{S\text{-}box}.$$

Here we designed $\mathcal{R}$ using an unaligned approach [BDKV21], similar to KECCAK, and treat the entire state as a three-dimensional cube. Specifically, non-linear S-box confusion and matrix diffusion operations act on each slice, while rotation acts on each lane, these components work together to achieve full state mixing. Next, we will introduce the criteria for selecting parameters for each component.

5.4.1 Choice of the MixSlice

The delay of the XOR operation for N bits, $x_0 \oplus x_1 \cdots x_{N-1}$, is equal to $\log_2 N$ multiplied by the delay of an XOR gate. It is most cost-effective to choose N as a power of 2. In this case, we choose $N = 4$. We maximize efficiency by combining all XOR operations, including **MixSlice** and **AC** operations. This can be achieved by commuting $\mathbf{LaneRotation}_1$ and **AC** since they are linear functions.

To design the **MixSlice**, we require a 16-by-16 invertible matrix with each row having a Hamming weight of 3. However, testing all about 2^{146} candidates is not feasible. Therefore, we decided to search for a matrix with desirable properties from the circulant matrices.

The diffusion of the **MixSlice** plays a crucial role in enhancing the resistance of $\mathcal{R}$ against differential and linear attacks. While the branch number is typically used to measure diffusion, all candidates in our case have a branch number of 4. However, it is important to note that the resistance against attacks can still vary, even with the same branch number. In order to identify a matrix such that $\mathcal{R}$ exhibit better properties, we calculated the outputs of each candidate for all possible inputs. Then we recorded the number of input-output pairs with specific Hamming weights and utilized the mapping $\mathcal{C}(w_i, w_o)$ to represent the count of pairs with input/output Hamming weights w_i and w_o respectively. Our objective is to select a matrix that can efficiently map a small number of active bits to multiple active bits, both for the matrix itself and its inverse. The matrix we have chosen possesses the following properties:

- If the input of the inverse of this matrix has a Hamming weight of 1, the output will have the highest possible Hamming weight among all matrices. In our specific case, the maximum Hamming weight is 9.
- For any other matrices, if the corresponding mapping C' differs from the mapping C of this matrix, there must exist values i and j such that

$$\begin{aligned}
&\mathcal{C}(w_i, w_o) = \mathcal{C}'(w_i, w_o), \forall w_i < i \text{ and } w_o \\
&\mathcal{C}(w_i, w_o) = \mathcal{C}'(w_i, w_o), \forall w_i = i \text{ and } w_o < j \\
&\mathcal{C}(w_i, w_o) < \mathcal{C}'(w_i, w_o), w_i = i \text{ and } w_o = j.
\end{aligned}$$

The first property ensures that the input difference is 9 when the output difference has only one active bit. The minimum number of active S-boxes for three-round differential trails can be increased to 13, resulting in the trail $9 \to 1 \to 3$. However, for other certain matrices, the minimum can only be 7, leading to the trail $3 \to 1 \to 3$. The second property states that when the input difference has fewer significant bits, this matrix has the lowest probability of producing an output difference with fewer significant bits compared to other matrices.

The linear properties depend on the transposition of the matrix. Since the matrix is a circulant matrix, its transposition also possesses the same properties as the matrix. Therefore, the linear resistance is similar to the differential resistance.

5.4.2 Choice of the S-boxes

Let us begin by examining some indicators related to the S-box. Given an S-box $\mathbf{S}$, we define $CarD1_{\mathbf{S}}$ as the number of trails in which a 1-bit input difference results in a 1-bit output difference. Similarly, $CarL1_{\mathbf{S}}$ represents the number of trails in which a 1-bit input active linear mask leads to a 1-bit active output linear mask. If a 1-bit active input difference (or mask) cannot propagate to a 1-bit active output, it is considered a good input. Conversely, if it can propagate, it is referred to as a bad input. Similarly, if a 1-bit active output difference (or mask) cannot originate from a 1-bit active input, it is considered a good output. Otherwise, it is called a bad output.

In the case of differential (or linear) analysis, we denote GI_D (or GI_L), GO_D (or GO_L), BI_D (or BI_L), and BO_D (or BO_L) as the sets of positions for the nonzero bits in the good inputs, good outputs, bad inputs, and bad outputs, respectively. Let GI, GO, BI, and BO represent the intersection of GI_D and GI_L, GO_D and GO_L, BI_D and BI_L, and BO_D and BO_L, respectively.

In recent years, there has been increasing concern about the impact of these indicators on the resistance of ciphers that solely rely on S-box and permutation operations in their round functions, such as `PRESENT` [BKL+07], `RECTANGLE` [ZBL+14], and `GIFT` [BPP+17], to differential and linear attacks.

It is worth noting that these indicators of the S-boxes also impact the resistance of the $\mathcal{R}$ against differential and linear attacks. This is because the active bits spread caused by the S-boxes on the slice can be further propagated throughout the cube through lane rotation operations.

To enhance the resistance of $\mathcal{R}$, we will utilize the properties of the chosen S-box **Sb**, which is $\mathtt{B}_9$ from the "Optimal BOGI-applicable" PXE classes as defined in Def.1. For more details, please refer to [KHSH20].

Definition 1 ([KHSH20])**.** A 4-bit S-box $\mathbf{S}$ is called an optimal BOGI-applicable S-box if it fulfills these four conditions: (a) $\mathbf{S}$ is bijective; (b) $\mathbf{S}$ is BOGI-applicable, i.e. the size sum of GI and GO is not less than 4; (c) the differential uniformity of $\mathbf{S}$ is 6; (d) the linearity of $\mathbf{S}$ is 8.

Regarding **Sb**, it can be easily confirmed that $GI = BO = \{0, 2\}$ and $GO = BI = \{1, 3\}$. This means that any bits in an even position of the slice are good inputs (resp. bad outputs), while any bits in an odd position of the slice are bad inputs (resp. good outputs). Since the matrix used in **MixSlice** is circulant, the analysis of the differential attack is similar to that of the linear attack. In this discussion, we will focus solely on the differential analysis.

It is straightforward to verify that **MixSlice** can map 1 bad output to 2 good inputs and 1 bad input, and map 1 good output to 1 good input and 2 bad inputs. Note that **LaneRotation** does not change the position in the slice. Therefore, even if a 1-bit bad input of the first round propagates to a 1-bit bad output after the S-box operation, the output of the S-box operation in the second round will have a minimum of 5 active bits. On the other hand, if the input is a 1-bit good input, the output of the S-box operation in the second round can also achieve at least 5 active bits.

Benefiting from the aforementioned properties, the minimum number of active S-boxes for $\mathcal{R}^3$ from a 1-bit active input can reach 19, and the trail of the number of active S-boxes is as follows: $1 \rightarrow 3 \rightarrow 15$. This leads to the possibility of achieving a minimum of 28 for 4-round trails. However, if we replace **Sb** with 4-bit low-latency S-boxes such as those used in `Orthros` or `Midori`, the lower bound reduces to 22. This suggests that in order to maintain the same level of security, there will exist at least one round gap between using **Sb** and these low-latency S-box.

We also used the Peigen platform [BGLS19] to assess the depth (Please referring to Definition 2) of 20 optimal PXE classes for BOGI applications. We followed the assumption of [BBI+15] that the depths of AND/OR, NAND/NOR, XOR/NXOR, and

NOT are estimated to be 1.5, 1, 2, and 0.5, respectively. We found that there is no S-box with depth less than 4. It is important to note that the minimum depth for 4-bit optimal S-boxes is 3.5 [BIL+21]. Finally, we chose $\mathtt{B}_9$ with depth 4 based on the fact that the probability of each differential (resp. linear) transactions from a bad input to a bad output for $\mathtt{B}_9$ is minimal.

We evaluated the latency of different 4-bit S-boxes, as shown in the Appendix Table 13. The variation in latency among the S-boxes is relatively minor, and the latency difference between **Sb** ($\mathtt{B}_9$) and lowest-latency S-boxes is only 0.04 *ns*. While the latency of a single round is approximately 0.4 *ns*. Besides, the 4-round trails of the cipher using $\mathtt{B}_9$ has more active S-boxes, at least 28, we consider it to be latency efficient.

Definition 2 (Depth [BBI+15, BIL+21])**.** The depth is defined as the sum of the sequential path delays of basic operations, namely AND, OR, NAND, NOR, XOR, NXOR and NOT.

5.4.3 Choice of the Offsets of LaneRotation

Due to the negligible delay of **LaneRotation** in hardware implementation, our focus should be on efficiently spreading the active bits across multiple S-boxes when determining the offsets. However, it is computationally unfeasible to test all possible **LaneRotation**. Therefore, we opt to randomly select the **LaneRotation** that fulfills the following conditions.

1. The offsets of $\mathbf{LaneRotation}_0$ and $\mathbf{LaneRotation}_1$ are mutually exclusive modulo l, where l is the length of the lane, and $l = 80$.

2. If an output linear mask has only two active bits located in different slices, the input linear mask of $\mathbf{MixSlice} \circ \mathbf{LaneRotation}_0$ must be located at least 5 rows.

3. If an input difference has only two active bits located in different slices, the output difference of $\mathbf{LaneRotation}_1 \circ \mathbf{MixSlice}$ must be located at least 5 rows.

4. In the three-round differential path, if there are 1 and 3 active S-boxes in the first round and second round respectively, then the offsets should guarantee that the minimum number of active S-boxes in the third round is at least 15.

5. In the three-round linear path, if there are 3 and 1 active S-boxes in the second and third round respectively, then the offsets should guarantee that the minimum number of active S-boxes in the first round is at least 15.

6. The offsets should guarantee that the IV attains 3-round full diffusion and the key achieves 4-round full diffusion.

It is challenging to search the combination of $\mathbf{LaneRotation}_0$ and $\mathbf{LaneRotation}_1$ that fulfills all these conditions. As a result, we adopt a three-step approach. Initially, we search for $\mathbf{LaneRotation}_0$ solutions that satisfy condition 1 and condition 2. Simultaneously, we search for $\mathbf{LaneRotation}_1$ solutions that fulfill condition 1 and condition 3. Subsequently, we merge the $\mathbf{LaneRotation}_0$ candidates and $\mathbf{LaneRotation}_1$ candidates, selecting combinations that meet condition 4 and condition 5. Finally, we choose the combination that satisfies condition 6 as the $\mathbf{LaneRotation}_0$ and $\mathbf{LaneRotation}_1$ from the candidate combinations.

As a result, the minimum number of active S-boxes for 4-round differential and linear trails is 28, which is the largest possible. See Table 14 and Table 15 for details.

5.4.4 Choice of the Round Constants

The round constants RC_i ($i \in \{1, \cdots, 19\}$) are derived from the fractional part of $\pi = 3.14159\cdots$, similar to `PRINCE` [BCG+12]. Initially, we selected the first 10,000 digits of the fractional part and then converted them from decimal to binary. Finally, we chose the first 2,880 bytes as $RC_1||\cdots||RC_{19}$, which can be found in Appendix B.

6 Security Analysis

We will first analyze the security of `Twinkle` through various cryptanalysis methods. Then, we will discuss the security of the WC-MAC. Furthermore, the security of the `Twinkle-AE` family depends on the security of the `Twinkle` and the WC-MAC. The security of the `Twinkle-PA` is also determined by the security of the PRF `Twinkle`.[1]

6.1 The Security of `Twinkle`

We will analyze `Twinkle` using various common cryptographic methods. In the memory encryption and pointer authentication scenario, the key is generated and securely stored within the processor. It is assumed that a typical adversary cannot access or modify the key. Therefore, the security of `Twinkle` against related-key attacks is not considered.

6.1.1 The Security of Even-Mansour

If the central permutation P in Even-Mansour construction (referring to Equation 5) is thoroughly random, the security bound is $\mathcal{O}(2^{\rho})$, where ρ is the block size. If $k^0 = k^1$, the bound will be reduced to $\mathcal{O}(2^{\rho/2})$ [EM97, Dae91, CLL+14]. In `Twinkle`, both k^0 and k^1 are derived from the ρ-bit key K, but k^0 is not equal to k^1. It implies that the `Twinkle`'s construction is bounded by $\mathcal{O}(2^{\rho/2})$, i.e., $\mathcal{O}(2^{640})$, which is far beyond security requirements of `Twinkle`.

6.1.2 Differential Attack

Differential attack [BS91] is one of the most powerful cryptanalysis methods. To search the differential trails effectively, it is often advantageous to transform it into a MILP problem [MWGP12, SHW+14] or Boolean satisfiability problem (SAT) and satisfiability modulo theories (SMT)[MP13]. By doing so, one can leverage a general-purpose solver to automatically determine the bound on the differential probability (DP) of the trails. Recently, Sun et al. [SWW21] explored the impact of the encoding method on the efficiency of the search and proposed a strategy to accelerate the search for the differential and linear characteristics based on SAT. Another way to searching for differential characteristics is through dedicated tools [DVA12, MDA17, MMGD22], which performs better on round functions involving bit shifts. Here, we employed both SAT method and dedicated tools for the differential analysis, and the results are listed in Table 5.

After the input expansion operation, the internal state contains 10 IVs. It implies when the IV difference has k active bits, the initial state difference will have $10k$ active bits. This property significantly increases in the minimum number of active S-boxes during the first few rounds. However, due to the large size of the state and the involvement of lane rotation operations in $\mathcal{R}$, it becomes challenging to compute compact bounds for additional rounds. Using the SAT method, we only obtained the bound of active S-boxes for the first 2-round differential trail as 43 with a weight greater than 80. The more precise weight bound of $\mathcal{R}^n$ is computed by the dedicated tool.

[1] The source code for the analysis is detailed in GitHub repository.

While we may not achieve superior results, the weights of the first 2-/6-/16-round trail are greater than 80, $80+58 = 138$ and $80+58\times3+9.4 = 263.4$, respectively, demonstrating that Twinkle could resist the differential attack within the security requirements. Moreover, the total bounds are calculated from the bounds of the segmented trails, which could not be naturally spliced together. We believe they possess ample security redundancy.

Table 5: The lower bounds of weight for `Twinkle`

Attacks	Construction	1	2	3	4
differential	first n rounds	25	>80	-	-
	$\mathcal{R}^n$	1.4	9.4	28.7	>58
linear	$\mathcal{R}^n$	2	8	28	60
	last $(n-0.5)$ rounds	4	14	-	-

6.1.3 Linear Attack

Computing the tight bounds for the weight of squared correlation (C^2) is also not a straightforward task due to the presence of a large internal state and bitwise permutation in the linear layer, and we were only able to obtain a four-round lower bound using dedicated tools, which is listed in Table 5.

According to Table 5, the weights of 9.5-/18.5-round linear trails are not less than $14+60\times2 = 134$, $60\times4+28 = 268$, respectively. For $\texttt{Twinkle}^{64}$, the lower bound of the number of active S-boxes in the last three-round trails is greater than 52 due to the limited output length of at most 64 bits. Therefore, we can conclude that $\texttt{Twinkle}^{64}$, $\texttt{Twinkle}^{128}$ and $\texttt{Twinkle}^{256}$ are sufficiently secure against linear attacks. Similar to differential analysis, the trails can not be naturally spliced together, and we are confident that it would be challenging to find a linear characteristic whose squared correlation is near the security bounds.

6.1.4 Integral Attack and Cube Attack

In [Tod15], Todo introduced the division property which is a generalization of the integral property. Following that, the bit-based division property [TM16] was proposed for refined integral construction. To search the integral distinguishers by off-the-shelf solvers, Xiang et al. [XZBL16] and Sun et al. [SWW17] modeled the propagation of the division property into mixed integer linear programming (MILP) and SAT (SMT), respectively. To analyze the division property of `Twinkle`, we described it into SAT models and solved the models using the open source solver CaDiCaL [Bie19].

From the results, we discovered that when all IV bits are active, the division trails could be found from initial state to any 5-round output. This implies that the algebraic degree of each output bit after 5 rounds, with respect to IV, almost reaches the maximum value of 128. This is because it is difficult to eliminate all terms that are divisible by $\prod_{i=0}^{127} IV[i]$ through XOR operations. Additionally, each bit of the initial state is in the form of $IV[i] \oplus k^0[j]$. Assuming that the cube set is $\{IV[i]\}_{i\in I}$, the algebraic degree with respect to key K of the cube sum after 5 rounds is expected to be at least $128-|I|$, where $|I|$ is the size of the set I. Therefore, we think that both integral attacks and cube attacks do not threaten the security of `Twinkle`.

6.1.5 Impossible Differential Cryptanalysis

The impossible difference analysis [BBS99] is a commonly used method for cryptanalysis. The number of rounds for full diffusion can be used to estimate the number of rounds for impossible difference distinguishers with a probability of 1. For `Twinkle`, any IV bits requires 3 rounds $\mathcal{R}$ for full diffusion, any internal bits require 4 rounds $\mathcal{R}$ for full diffusion, and 3 rounds $\mathcal{R}^{-1}$ for full diffusion, referring to Table 6. Therefore, it's expected that `Twinkle` does not have a 6 round distinguisher with a probability of 1 including the initial round, and a usual 7 round distinguisher.

Furthermore, it is challenging to extend more rounds due to the low probability of differential trails from the initial round, as indicated in Table 5. Additionally, since at least 128 bits (1216 bits for $\texttt{Twinkle}^{64}$) before the operation $\mathcal{F}_O$ are unknown, and need to be guessed to determine the difference, we believe that the impossible differential attack does not pose a threat to all versions of `Twinkle`.

Table 6: Upper bounds for the number of influenced bits

Rounds	internal bits		IV bits
	$\mathcal{R}$	$\mathcal{R}^{-1}$	$\mathcal{R}$
0	1	1	1
1	12	36	12
2	144	888	101
3	1004	**1280**	**128**
4	**1280**	**1280**	**128**

6.1.6 Truncated Differential Attack

It is crucial to analyze the resistance of `Twinkle` against truncated differential attacks [Knu95]. These attacks can be executed by collecting differential trails with the same truncated input and output difference or by directly searching for high-probability truncated differential trails.

Based on the results of the differential cryptanalysis in Table 5, the weights of the 5-/9.5-/18.5- round differential trails are greater than $80 + 28.7 = 108.7$, $80 + 58 \times 2 = 196$, and $80 + 58 \times 4 + 1.4 = 313.4$, respectively. This implies that the adversary would need to identify at least $2^{44.7}$, 2^{68}, and $2^{57.4}$ high-probability trails with the same truncated input and output difference to attack $\texttt{Twinkle}^{64}$, $\texttt{Twinkle}^{128}$, and $\texttt{Twinkle}^{256}$, respectively. Moreover, more trails would be required due to the overestimated probability of differential trails.

To search for a high-probability "long" truncated differential trail, we construct it by combining a normal differential trail with a "short" truncated trail. (The terms "long" and "short" are used here for distinction and do not imply any specific length.) We assume, without loss of generality, that the first round of the "short" truncated trail exhibits truncated differential propagation. The linear structure (referring to Definition 3) can describe the truncated differential propagation of an S-box. Using the PEIGEN platform, we computed the linear structure of the S-box **Sb**, which is listed in Table 7.

Definition 3 (Linear structures of an S-box [Eve88, Lai95, Dub01, BGLS19])**.** A linear structure of an S-box $\mathbf{S}$: $\mathbb{F}_2^n \to \mathbb{F}_2^n$ is a triple $(\lambda, a, c) \in \mathbf{F}_2^n \times \mathbf{F}_2^n \times \mathbf{F}_2$ such that

$$\lambda \cdot \mathbf{S}(x) \oplus \lambda \cdot \mathbf{S}(x \oplus a) = c \quad \text{for } \forall x \in \mathbb{F}_2^n.$$

The process of the attack is as follows.

1. **Identify the "short" truncated trail.**

   The uncertainty in active S-box differential propagation, whether probabilistic normal or truncated, complicates the search for the optimal path. It is established that if a bit in the input difference is unknown, the related output differential bit will also be unknown. Under the assumption of the "short" truncated trail, at least one unknown bit exists in the first S-layer output difference. By setting the output

Table 7: The linear structures of **Sb**

λ	a	c
0001	0101	1
0001	1000	1
0001	1101	0
0100	0010	1
0100	0101	1
0100	0111	0
0101	0101	0
0101	1010	1
0101	1111	1

difference of the 1st-round S-layer to have only one unknown bit, denoted by U, we can determine the maximum possible unaffected bits for the last $i/i+0.5$-round output difference in different versions of `Twinkle`. The details are provided in Table 8, where the last i-round for $\texttt{Twinkle}^{64}$ and the last $i+0.5$-round for $\texttt{Twinkle}^{128}$ and $\texttt{Twinkle}^{256}$ (both containing $\mathcal{F}_O$ operation, see Section 3.5). The cost of fixing the bits affected by U is typically higher than fixing U itself. Therefore, a good "short" truncated difference path should not exceed 4 rounds for $\texttt{Twinkle}^{64}$, and 3.5 rounds for $\texttt{Twinkle}^{128}$ and $\texttt{Twinkle}^{256}$ heuristically.

As a result, we have identified a 3.5-round truncated differential trail with a probability of 1, where four bits in the output differential are fixed at 0. However, there is no 4-round truncated difference trail with a probability of 1. This is because in the case where the unknown bit of the 1st-round S-layer's output difference is only located at $(2,0,*)$ or $(2,1,*)$ or $(1,2,*)$, the output difference for a last 4-round trail has only 1 known bit located at $(3,1,*)$, $(2,1,*)$, $(0,0,*)$, respectively. And any truncated differential propagation of **Sb** with a probability of 1 results in at least 2 unknown bits in the output. This also implies that the first round of a 4-round "short" truncated trail can have only one S-box involved in truncated differential propagation. Instead, we have found a truncated difference trail with a probability of $2^{-7.4}$, where one bit in the output difference is fixed at 0. Refer to Table 16 and Table 17 for more details.

2. **Combine with normal differential trail.**

   For $\texttt{Twinkle}^{64}$, the probability weight of any normal first 2-round differential trail is already over 80, making it impractical to combine with a "short" truncated trail within 3 rounds. Expanding a 4-round "short" trail to a full-round "long" trail is challenging due to the presence of $\mathcal{F}_I$. However, the complexity of an attack against $\texttt{Twinkle}^{64}$ remains greater than $\mathcal{O}(2^{64})$ due to the reduction in probability weight of the first 2-round trail by at most 3 (only 1 S-box's differential propagation transforms to truncated differential propagation when considering the 2nd round as the 1st round of a 4-round "short" truncated trail). This suggests that $\texttt{Twinkle}^{64}$ is resistant to truncated differential attacks.

   For $\texttt{Twinkle}^{128}$ and $\texttt{Twinkle}^{256}$, the probability weight of any initial 6-round, 15-round differential trails with known input and output exceeds 138/255.4. This makes it impossible for an adversary to distinguish any "long" truncated differential trail of $\texttt{Twinkle}^{128}/\texttt{Twinkle}^{256}$ within a computational complexity of $\mathcal{O}(2^{128})/\mathcal{O}(2^{256})$.

Table 8: The maximum unaffected bits of last $i + 0.5$-/i-round output difference when only 1 unknown bit in the output difference of 1st-round S-layer

#rounds	#unaffected output difference over 1152 bits	#rounds	#unaffected output difference over 64 bits
1.5	1011	1	64
2.5	833	2	64
3.5	106	3	44
4.5	0	4	1
-	-	5	0

6.1.7 Invariant Attacks

In [BCLR17], Beierle et al. propose a method of proving resistance against invariant attacks according to the linear layer and the round constants, which covers the invariant subspace attack [LAAZ11, LMR15, GJN+16] and nonlinear invariant attack [TLS16]. Let D be a set of known differences between round constants, i.e., a subset of all $(RC_i \oplus RC_j)$. The smallest linear subspace invariant under L is defined as follows:

$$W_L(D) := \sum_{c \in D} \langle L^i(c), i \geq 0 \rangle = \sum_{c \in D} W_L(c),$$

where L is the linear layer, and $L = \mathbf{LaneRotation_1} \circ \mathbf{MixSlice} \circ \mathbf{LaneRotation_0}$ in our case. Then $W_L(D)$ could be computed as in Algorithm 1. Suppose that the dimension of $W_L(D)$ is at least $\rho - 1$, where ρ is the block size. Then there is no non-trivial invariant of the S-box layer to cover $W_L(D)$, unless the S-box layer has a component of degree 1.

Algorithm 1 Computing $W_L(D)$

Require: list of differences D, linear layer L as a matrix
Ensure: the subspace $W_L(D)$
$R \leftarrow$ an empty list
$k \leftarrow$ the multiplicative order of L
for all c in D **do**
  for all j from 0 to $k - 1$ **do**
    Add $L^j(c)$ into R
  end for
end for
return the span of R

For the difference of two consecutive round constants $c = RC_i \oplus RC_{i+1}$ of `Twinkle`, we found that the dimension of $W_L(c)$ will be at least 1279 for most c (see Table 9 for details), attributed to the strong diffusion and unaligned property of L. It implies that for any $c = RC_i \oplus RC_{i+1}$ resulting in $\dim(W_L(c)) \geq 1279$, there is no non-trivial invariant for both S-Box layer and linear layer of $\mathcal{R}_{i+1} \circ \mathcal{R}_i$. Therefore, the invariant attacks do not threaten `Twinkle`.

Table 9: The dimensions of $W_L(RC_i \oplus RC_{i+1})$

i	$W_L(RC_i \oplus RC_{i+1})$	i	$W_L(RC_i \oplus RC_{i+1})$
1	1280	10	1279
2	1280	11	1279
3	1280	12	1280

4	1280	13	1278
5	1280	14	1278
6	1280	15	1276
7	1278	16	1280
8	1280	17	1280
9	1280	18	1279

6.1.8 Meet-in-the-Middle Attack

For `Twinkle`^{128} and `Twinkle`^{256}, performing a meet-in-the-middle (MITM) attack with more than 7 rounds is challenging due to the key being fully mixed in 4 rounds forward and 3 rounds reverse. Moreover, up to 2^{1280} key space and 128-bit hidden output also enhance resistance against MITM attacks.

For `Twinkle`^{64}, the key size involved after 2 rounds already exceeds 128 bits, referring to Table 6. And the 64-bit output represents only a fraction of the 1280-bit state, requiring an attacker to guess an impractically large number of bits for a successful MITM attack. Therefore, `Twinkle`^{64} is secure against MITM attacks with a security requirement of 64 bits.

6.1.9 Guess and Determine Attack

We recall `Twinkle` $= \mathcal{F}_O(P(\mathcal{F}_I(IV) \oplus k^0) \oplus k^1)$, where $P = \mathcal{R}_R \circ \cdots \circ \mathcal{R}_1$ as shown in Figure 3. First, we explore the possibility that an adversary could attempt to determine the input of $\mathcal{F}_O$ to recover the key. An adversary would need to guess at least 128 bits to determine the 1280-bit input of $\mathcal{F}_O$, due to the leak of $\mathcal{F}_O$ at most 1152 bits. The guess results in a complexity of $\mathcal{O}(2^{128})$ or higher, exceeding the security bounds of `Twinkle`^{128} and `Twinkle`^{64}. For `Twinkle`^{256}, the XOR operation with k^1 complicates key recovery, transforming it into a problem of recovering the key of a 1-round Even-Mansour structure. When P is a pseudo-random permutation function, the Even-Mansour structure can provide 640-bit security ([EM97, Dut20]). Moreover, distinguishing $P = \mathcal{R}^{18.5}$ from a PRP with complexity less than $\mathcal{O}(2^{128})$ is challenging, making key recovery infeasible using this method.

Alternatively, the adversary also could potentially guess some bits of k^0 and k^1, and perform forward and backward calculations with the IV, output and guessed key to establish equations on the unguessed key bits in the middle rounds. However, each bit of the state after three rounds involves approximately a 1000-bit key and has a degree of about 27. Besides, for only 5-round `Twinkle`^{64}, it is difficult to compute backward due to only 64-bit output . Therefore, we believe this approach does not pose a significant threat to the security of `Twinkle`.

6.2 The Security of `Twinkle-AE`

`Twinkle-AE` family could provide 128-, 128-, 256-bit confidentiality security and 64-, 128-, 128-bit integrity security for the plaintext. Each pair of the IV and key should only be used for processing one plaintext to ensure that the `Twinkle-AE` family meets the security goals. Additionally, the decrypted plaintext should only be released if the tag verification is successful.

The confidentiality of `Twinkle-AE` relies on the security of `Twinkle`, while the integrity of `Twinkle-AE` is dependent on the WC-MAC. As the security of `Twinkle` has already been discussed, our attention will now shift to the security of WC-MAC.

6.2.1 The Security of the WC-MAC

Because `Twinkle`^{128} and `Twinkle`^{256} have at least 128 bits of security level, so the at most 128-bit output for authentication is uniformly random. Therefore, for 64-/128-bit tag, the success probability of 1 forgery by the adversary is at most $2^{-64}/2^{-128}$, which is equivalent to "blind guess".

6.3 The Security of `Twinkle-PA`

`Twinkle-PA` offers 64-bit security against offline attacks, and provides t-bit security against online attacks, where t represents the length of the tag. In the offline attack, we assume that the adversary can gather the tuples of the pointer, context, and corresponding tag, and utilize this information to recover the MAC key offline. In the online attack, as for each key, the adversary has only one opportunity to manipulate the context and pointer in order to deceive the authentication process. Once failed, the system will force a reset with new keys.

6.3.1 Offline Attacks

To recover the MAC key, potential offline attacks include differential attack, linear attack, cube attack, integral attack and so on. The security against these attacks is guaranteed by the security of `Twinkle`^{64}.

6.3.2 Online Attacks

Differential Forge Attack. By the differential analysis of `Twinkle`^{64}, it is unfeasible to forge a pair of pointers and context that can successfully pass the verification process by studying the differential trail.

7 Hardware Evaluation

Settings. Our goal is to develop low latency schemes, so we will be using the fully unrolled circuit in the hardware implementation for performance evaluation. Due to limitations in the experimental environment, we will be using the FreePDK45 kit, an open-source generic process design kit, for our tests.

To ensure a fair comparison, we first obtained the delays of all ciphers at a low clock frequency. Next, we constrained the total signal delay for each design to 85%, 80%, and 75% of its delay obtained at the low clock frequency. Finally, we calculated the minimum latency for each design.

Categories of candidate designs. We conducted experiments to assess the hardware performance of our designs, and categorized the candidates for comparison into three groups:

- **PRFs/PRPs.** We compared the performance of the `Twinkle` PRF with various low-latency designs, including block ciphers (PRPs) such as `PRINCE` [BCG+12] `QARMA` [Ava17], `QARMAv2` [ABD+23] family, `AES` [DR98] and a PRF `Orthros` [BIL+21]. Notably, the recently introduced PRF `Gleeok` [ABC+24] has garnered attention; however, due to time constraints, we were unable to implement and assess its performance firsthand. Nevertheless, we offer our estimation for consideration.

  Concerning latency, the 12-round `Gleeok` PRF shares a similar architectural foundation with `Orthros`, and the latency metrics reported in the `Gleeok` paper are comparable to those of `Orthros`—358.52 ps for `Gleeok-128` versus 351.55 ps for

Orthros. In terms of area, Gleeok is anticipated to demand approximately 1.5 times the area of Orthros, attributable to Gleeok's utilization of three branches as opposed to Orthros's two.

- **AE schemes.** We compared the performance of the Twinkle-AE scheme with Ascon [DEMS21] and the authentication encryption scheme that embeds the above PRPs and PRFs. It is important to note that the security bounds provided by these PRPs and PRFs are different. For a fair comparison, we only considered the authenticated encryption schemes for these functions with same security (including trade-off security bound), where integrity level is not less than 64 bits. Based on the goal of minimizing delay, the combination of CTR mode and WC-MAC [Gue16] is suitable for block ciphers, while the Flat-ΘCB scheme [IMO+22] is suitable for tweakable block cipher. Both schemes have authentication security bounded by $\mathcal{O}(2^{b/2})$, where b is the block size. This means that for a 64-/128-bit block cipher with these schemes, 64-/128-bit authentication security cannot be provided, respectively. To achieve 64-/128-bit authentication security, only beyond-birthday-bound MAC could be used, but this would worsen latency performance and make it uncompetitive. Therefore, we did not evaluate the performance of authentication encryption scheme for the 64-/128-bit block ciphers when more than 64-/128-bit integrity required, respectively.
- **Pointer authentication schemes.** We also evaluated current pointer authentication schemes, including our Twinkle-PA, QARMA-64 [Ava17], QARMAv2-64 [ABD+23] family and BipBip [BDD+23].

7.1 Results

The evaluation of PRPs and PRFs. Table 10 shows the performance of PRPs and PRFs, highlighting the exceptional latency of our $\texttt{Twinkle}^{64}$, $\texttt{Twinkle}^{128}$, and $\texttt{Twinkle}^{256}$, with delays of 2.04 *ns*, 3.83 *ns*, and 7.34 *ns*, respectively.

$\texttt{Twinkle}^{64}$ is used in Twinkle-PA which has a lower security level with reduce round number. Hence the minimal latency is expected.

At the 128-bit security level, $\texttt{Twinkle}^{128}$'s has the lowest latency, 3.83 *ns* followed by Orthros, another PRF, which achieves 4.34 *ns*.

Overall, we observe that the designs with asymmetric delay components potentially show better delay at the same security level.

Additionally, PRINCE has the lowest delay among the 64-bit block ciphers, but the tweakable block cipher QARMA is more flexible for use in memory encryption applications.

The evaluation of AE schemes. The results of each authentication and encryption scheme are shown in Table 11. We categorize the schemes based on their security levels. The majority of the candidates fall into the first category, offering 128-bit security for confidentiality and 64-bit security for integrity. Experimental data reveal that $\texttt{Twinkle}^{128}$ exhibits the lowest latency. When compared to the AES-CTR with WC-MAC, employed in the Intel SGX solution, Twinkle-AE achieves a 60.8% improvement in encryption latency and a 53.1% reduction in decryption latency. In terms of area efficiency, QARMA is the most area-efficient in the 512-bit cache line configuration, closely followed by $\texttt{Twinkle}^{128}$. For the 1024-bit cache line, $\texttt{Twinkle}^{128}$ has the highest area utilization due to its compact design requiring only one block for authenticated encryption. In comparison, QARMA, AES, and Orthros need up to nine blocks for the same output. Additionally, as a PRF, Orthros has a low output-to-state ratio of 0.5, while Twinkle has a higher ratio of 0.9, making it more area-efficient. This suggests that a dedicated design with a tailored-size state could be more advantageous compared to PRFs with XORing multiple blocks.

Table 10: Results for PRPs and PRFs

PRPs/PRFs	Max Output Size (bits)	Security Level	Delay (ns)	Area (μm^2)	Throughput ($Gbps$)
`PRINCE`	64	$D \le (2^n), T \ge (2^{127-n})$	4.73	9096.4	13.5
`QARMA-64-`σ_0	64	$D \le (2^n), T \ge (2^{128-n-\epsilon})$	5.57	14543.5	11.5
`QARMA-64-`σ_1	64	$D \le (2^n), T \ge (2^{128-n-\epsilon})$	5.85	15544.2	10.9
`QARMA-64-`σ_2	64	$D \le (2^n), T \ge (2^{128-n-\epsilon})$	6.11	16673.9	10.5
`QARMAv2`$_7$`-64`	64	$D \le (2^{56}), T \ge (2^{128-\epsilon})$	5.65	15498.2	11.3
`QARMA-128-`σ_0	128	$D \le (2^n), T \ge (2^{256-n-\epsilon})$	8.75	37315.0	14.6
`QARMA-128-`σ_1	128	$D \le (2^n), T \ge (2^{256-n-\epsilon})$	9.19	42914.6	13.9
`QARMA-128-`σ_2	128	$D \le (2^n), T \ge (2^{256-n-\epsilon})$	9.63	43199.2	13.3
`QARMAv2`$_9$`-128`	128	$D \le (2^{80}), T \ge (2^{128-\epsilon})$	7.10	38550.8	18.0
`QARMAv2`$_{11}$`-128`	128	$D \le (2^{80}), T \ge (2^{192-\epsilon})$	8.86	43820.6	14.4
`QARMAv2`$_{13}$`-128`	128	$D \le (2^{80}), T \ge (2^{256-\epsilon})$	10.27	51624.2	12.5
`AES`	128	128 bits	9.78	115935.6	13.1
`Orthros`	128	128 bits	4.34	34346.2	29.5
`Twinkle`^{64}	**64**	64 bits	**2.04**	**62990.9**	**31.4**
`Twinkle`^{128}	**1152**	128 bits	**3.83**	**120004.8**	**300.8**
`Twinkle`^{256}	**1152**	256 bits	**7.34**	**219445.7**	**156.9**

Ascon's higher latency can primarily be attributed to its serial structure, necessitating multiple primitive calls for processing 512/1024-bit messages. Nonetheless, the area footprint of Ascon remains relatively small, aligning with expectations.

Table 11: Results for authentication and encryption/verification and decryption process

confidentiality/ integrity	Schemes	AE Delay (ns)	VD Delay (ns)	Area (μm^2) 512-bit CL	1024-bit CL
128-bit/64-bit	`QARMAv2`$_9$`-128`*	7.10	7.10	192754.0	346957.2
	`AES`†	9.78	11.2	667879.6	1219824.0
	`Orthros`‡	4.34	5.76	259932.6	485519.0
	`Twinkle`^{128}‡	**3.83**	**5.25**	**208206.4**	**296408.0**
128-bit/128-bit	`Orthros`‡	4.34	6.42	338068.2	641790.2
	Ascon	27.3	27.3	163237.5	-
	`Twinkle`^{128}‡	**3.83**	**5.91**	**286342.0**	**452679.2**
256-bit/128-bit	**`Twinkle`^{256}‡**	**7.34**	**9.42**	**385782.9**	**552120.1**

* `Flat-`Θ`CB` Scheme: AE Delay = 1 TBC; VD Delay = 1 TBC; Area = $(c/o + 1)$ TBC.
† CTR + WC-MAC: AE Delay = 1 BC; VD Delay = 1 BC + 1 multi.; Area = $(c/o + 1)$ TBC + c/t multi..
‡ Stream + WC-MAC: AE Delay = 1 PRF; VD Delay = 1 PRF + 1 multi.; Area: $(\lceil c/o \rceil + 1)$ PRF + c/t multi..
t: the integrity level in bits; c: cache line size; o: max output size; multi.: multiply in $\mathbb{F}_2^t$. The latency and area of few XORs are ignored here.

The evaluation of pointer authentication. Table 12 shows that `Twinkle-PA` achieves a low latency of only 2.04 ns, while occupying an area of 62990.9 μm^2. It is important to note that in chips using the `Twinkle-AE` family, by reutilizing the existing partial circuitry, the incremental area needed for `Twinkle-PA` is confined to just a few additional XOR gates. This efficient reuse significantly minimizes the extra layout space required for `Twinkle-PA`.

Upon comparison, `Twinkle-PA`'s latency is found to be only 36.6% of that of the `QARMA-64` family, which is utilized in the ARMv8.3-A ISA extensions for pointer authentication. Furthermore, `Twinkle-PA`'s latency is at most 57.9% of the latency observed in the newer `QARMAv2-64` family variants. In the case of `BipBip`, the decryption latency stands at 4.17 ns, which is double that of `Twinkle-PA`. This supports the expectation that a dedicated MAC with a large state and a one-way function would offer a significant performance advantage in terms of latency.

Table 12: Hardware performances for pointer authentication

Cipher	Delay (ns)	Area (μm^2)
QARMA-64-σ_0	5.57	14543.5
QARMA-64-σ_1	5.85	15544.2
QARMA-64-σ_2	6.11	16673.9
QARMAv2_4-64-σ_0	3.52	8898.2
QARMAv2_6-64-σ_0	4.88	12845.1
QARMAv2_4-64	3.59	9377.8
QARMAv2_6-64	4.99	13475.8
BipBip (Dec)	4.17	6721.3
Twinkle-PA	**2.04**	**62990.9**

Discussion. Although the evaluation results of different cell libraries may vary, we believe that Twinkle can achieve low-latency performance due to its circuit depth of only $8r + 6$ for the r-round Twinkle. We will open-source the hardware implementation for researchers to test under different cell libraries.

7.2 Discussion on protected implementations

In scenarios where protected implementations are essential, our design can readily incorporate common side-channel attack (SCA) protection methods, such as masking. This integration is feasible due to our use of a bit-sliced S-Box implementation with minimal logic complexity.

Nevertheless, we argue that protected implementations may not be necessary in this context for several reasons. Firstly, low-latency ciphers, which are inherently sensitive to performance degradation, may find it challenging to manage the extra burden that such protected implementations impose in real-world scenarios. Secondly, memory encryption engines, when integrated into high-end chips for the purpose of safeguarding sensitive data, are typically fortified with comprehensive countermeasures to thwart physical attacks. Additionally, the complexity of executing invasive attacks at the chip level is substantial. It demands a high level of expertise, considerable resources, and a significant time investment to execute successfully. This complexity often acts as a deterrent, further reducing the necessity for such protected implementations in these situations.

8 Conclusion

In this study, we have introduced the low-latency PRFs known as Twinkle. Building upon these PRFs, we have developed the dedicated low-latency authenticated encryption scheme Twinkle-AE and the pointer authentication algorithm Twinkle-PA. A comprehensive security evaluation was carried out for both the PRFs and the aforementioned schemes, assessing their resilience against a range of common attacks. Our cryptanalysis to date suggests that these designs meet their intended security levels.

The development of Twinkle involved the implementation of novel design strategies, specifically aimed at achieving low latency. These strategies were particularly focused on scenarios where the plaintext size is equivalent to the cache line size. Subsequent hardware evaluations confirmed that all variants within the Twinkle family effectively met our low-latency objectives, thereby endorsing the efficacy of our design approaches.

Looking forward, it presents an interesting avenue for future research to delve into additional low-latency design strategies, especially those tailored for specific use cases and scenarios. This exploration could further enhance the efficiency and applicability of low-latency cryptographic solutions in various real applications.

References

[ABC+24] Ravi Anand, Subhadeep Banik, Andrea Caforio, Tatsuya Ishikawa, Takanori Isobe, Fukang Liu, Kazuhiko Minematsu, Mostafizar Rahman, and Kosei Sakamoto. Gleeok: A family of low-latency prfs and its applications to authenticated encryption. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2024(2):545–587, Mar. 2024. URL: https://tches.iacr.org/index.php/TCHES/article/view/11439, doi:10.46586/tches.v2024.i2.545-587.

[ABD+23] Roberto Avanzi, Subhadeep Banik, Orr Dunkelman, Maria Eichlseder, Shibam Ghosh, Marcel Nageler, and Francesco Regazzoni. The qarmav2 family of tweakable block ciphers. *IACR Transactions on Symmetric Cryptology*, 2023(3):25–73, Sep. 2023. URL: https://tosc.iacr.org/index.php/ToSC/article/view/11184, doi:10.46586/tosc.v2023.i3.25-73.

[AMD19] AMD. Secure encrypted virtualization (sev), 2019. URL: https://developer.amd.com/sev/.

[ARM21] ARM. Arm CCA Security Model, August 2021. Rev 1.0, Document Number DEN0096.

[Ava17] Roberto Avanzi. The QARMA block cipher family. *IACR Trans. Symm. Cryptol.*, 2017(1):4–44, 2017. doi:10.13154/tosc.v2017.i1.4-44.

[Ava22] Roberto Maria Avanzi. Cryptographic protection of random access memory: How inconspicuous can hardening against the most powerful adversaries be? *Proceedings of the 2022 on Cloud Computing Security Workshop*, 2022. URL: https://api.semanticscholar.org/CorpusID:253187590.

[BBI+15] Subhadeep Banik, Andrey Bogdanov, Takanori Isobe, Kyoji Shibutani, Harunaga Hiwatari, Toru Akishita, and Francesco Regazzoni. Midori: A block cipher for low energy. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part II*, volume 9453 of *LNCS*, pages 411–436. Springer, Heidelberg, November / December 2015. doi:10.1007/978-3-662-48800-3_17.

[BBS99] Eli Biham, Alex Biryukov, and Adi Shamir. Cryptanalysis of Skipjack reduced to 31 rounds using impossible differentials. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 12–23. Springer, Heidelberg, May 1999. doi:10.1007/3-540-48910-X_2.

[BCG+12] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knežević, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçin. PRINCE - A low-latency block cipher for pervasive computing applications - extended abstract. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 208–225. Springer, Heidelberg, December 2012. doi:10.1007/978-3-642-34961-4_14.

[BCLR17] Christof Beierle, Anne Canteaut, Gregor Leander, and Yann Rotella. Proving resistance against invariant attacks: How to choose the round constants. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 647–678. Springer, Heidelberg, August 2017. doi:10.1007/978-3-319-63715-0_22.

[BDD+23] Yanis Belkheyar, Joan Daemen, Christoph Dobraunig, Santosh Ghosh, and Shahram Rasoolzadeh. Bipbip: A low-latency tweakable block cipher with small dimensions. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2023(1):326–368, 2023. doi:10.46586/tches.v2023.i1.326-368.

[BDKV21] Nicolas Bordes, Joan Daemen, Daniël Kuijsters, and Gilles Van Assche. Thinking outside the superbox. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part III*, volume 12827 of *LNCS*, pages 337–367, Virtual Event, August 2021. Springer, Heidelberg. doi:10.1007/978-3-030-84252-9_12.

[BDPA13] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Keccak. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 313–314. Springer, Heidelberg, May 2013. doi:10.1007/978-3-642-38348-9_19.

[BEK+20] Dusan Bozilov, Maria Eichlseder, Miroslav Knezevic, Baptiste Lambin, Gregor Leander, Thorben Moos, Ventzislav Nikov, Shahram Rasoolzadeh, Yosuke Todo, and Friedrich Wiemer. PRINCEv2 - more security for (almost) no overhead. In Orr Dunkelman, Michael J. Jacobson Jr., and Colin O'Flynn, editors, *Selected Areas in Cryptography - SAC 2020 - 27th International Conference, Halifax, NS, Canada (Virtual Event), October 21-23, 2020, Revised Selected Papers*, volume 12804 of *Lecture Notes in Computer Science*, pages 483–511. Springer, 2020. doi:10.1007/978-3-030-81652-0_19.

[Ber05] Daniel J. Bernstein. Stronger security bounds for Wegman-Carter-Shoup authenticators. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 164–180. Springer, Heidelberg, May 2005. doi:10.1007/11426639_10.

[BGLS19] Zhenzhen Bao, Jian Guo, San Ling, and Yu Sasaki. Peigen – a platform for evaluation, implementation, and generation of S-boxes. *IACR Trans. Symm. Cryptol.*, 2019(1):330–394, 2019. doi:10.13154/tosc.v2019.i1.330-394.

[Bie19] Armin Biere. Cadical at the sat race 2019, 2019.

[BIL+21] Subhadeep Banik, Takanori Isobe, Fukang Liu, Kazuhiko Minematsu, and Kosei Sakamoto. Orthros: A low-latency PRF. *IACR Trans. Symm. Cryptol.*, 2021(1):37–77, 2021. doi:10.46586/tosc.v2021.i1.37-77.

[BJK+16] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY family of block ciphers and its low-latency variant MANTIS. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part II*, volume 9815 of *LNCS*, pages 123–153. Springer, Heidelberg, August 2016. doi:10.1007/978-3-662-53008-5_5.

[BKL+07] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe. Present: An ultra-lightweight block cipher. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007*, pages 450–466, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. doi:10.1007/978-3-540-74735-2_31,.

[BKR98] Mihir Bellare, Ted Krovetz, and Phillip Rogaway. Luby-Rackoff backwards: Increasing security by making block ciphers non-invertible. In Kaisa Nyberg, editor, *EUROCRYPT'98*, volume 1403 of *LNCS*, pages 266–280. Springer, Heidelberg, May / June 1998. doi:10.1007/BFb0054132.

[BPH15] Andrew Baumann, Marcus Peinado, and Galen C. Hunt. Shielding applications from an untrusted cloud with haven. *ACM Trans. Comput. Syst.*, 33(3):8:1–8:26, 2015. doi:10.1145/2799647.

[BPP+17] Subhadeep Banik, Sumit Kumar Pandey, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, and Yosuke Todo. GIFT: A small present - towards reaching the limit of lightweight encryption. In Wieland Fischer and Naofumi Homma, editors, *CHES 2017*, volume 10529 of *LNCS*, pages 321–345. Springer, Heidelberg, September 2017. doi:10.1007/978-3-319-66787-4_16.

[BS91] Eli Biham and Adi Shamir. Differential cryptanalysis of DES-like cryptosystems. In Alfred J. Menezes and Scott A. Vanstone, editors, *CRYPTO'90*, volume 537 of *LNCS*, pages 2–21. Springer, Heidelberg, August 1991. doi:10.1007/3-540-38424-3_1.

[CGL+23] Federico Canale, Tim Güneysu, Gregor Leander, Jan Philipp Thoma, Yosuke Todo, and Rei Ueno. SCARF – a Low-Latency block cipher for secure Cache-Randomization. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 1937–1954, Anaheim, CA, August 2023. USENIX Association. URL: https://www.usenix.org/conference/usenixsecurity23/presentation/canale.

[CLL+14] Shan Chen, Rodolphe Lampe, Jooyoung Lee, Yannick Seurin, and John P. Steinberger. Minimizing the two-round Even-Mansour cipher. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 39–56. Springer, Heidelberg, August 2014. doi:10.1007/978-3-662-44371-2_3.

[com] The CAESAR committee. Caesar: competition for authenticated encryption: security, applicability, and robustness. https://competitions.cr.yp.to/caesar-submissions.html. 2014.

[CS16] Benoît Cogliati and Yannick Seurin. EWCDM: An efficient, beyond-birthday secure, nonce-misuse resistant MAC. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 121–149. Springer, Heidelberg, August 2016. doi:10.1007/978-3-662-53018-4_5.

[Dae91] Joan Daemen. Limitations of the even-mansour construction. In *Advances in Cryptology - ASIACRYPT '91, International Conference on the Theory and Applications of Cryptology, Fujiyoshida, Japan, November 11-14, 1991, Proceedings*, volume 739 of *Lecture Notes in Computer Science*, pages 495–498. Springer, 1991. doi:10.1007/3-540-57332-1_46.

[DEMS21] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1.2: Lightweight authenticated encryption and hashing. *Journal of Cryptology*, 34(3):33, July 2021. doi:10.1007/s00145-021-09398-9.

[DLT+23] Sen Deng, Mengyuan Li, Yining Tang, Shuai Wang, Shoumeng Yan, and Yinqian Zhang. Cipherh: Automated detection of ciphertext side-channel vulnerabilities in cryptographic implementations. In Joseph A. Calandrino and Carmela Troncoso, editors, *32nd USENIX Security Symposium, USENIX*

Security 2023, Anaheim, CA, USA, August 9-11, 2023. USENIX Association, 2023. URL: https://www.usenix.org/conference/usenixsecurity23/presentation/deng-sen.

[DR98] Joan Daemen and Vincent Rijmen. The block cipher rijndael. In Jean-Jacques Quisquater and Bruce Schneier, editors, *Smart Card Research and Applications, This International Conference, CARDIS '98, Louvain-la-Neuve, Belgium, September 14-16, 1998, Proceedings*, volume 1820 of *Lecture Notes in Computer Science*, pages 277–284. Springer, 1998. doi:10.1007/10721064_26.

[Dub01] Sylvie Dubuc. Characterization of linear structures. *Des. Codes Cryptography*, 22:33–45, 01 2001. doi:10.1023/A:1008399109102.

[Dut20] Avijit Dutta. Minimizing the two-round tweakable Even-Mansour cipher. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part I*, volume 12491 of *LNCS*, pages 601–629. Springer, Heidelberg, December 2020. doi:10.1007/978-3-030-64837-4_20.

[DVA12] Joan Daemen and Gilles Van Assche. Differential propagation analysis of keccak. In Anne Canteaut, editor, *Fast Software Encryption*, pages 422–441, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. doi:10.1007/978-3-642-34047-5_24.

[EM93] Shimon Even and Yishay Mansour. A construction of a cipher from a single pseudorandom permutation. In Hideki Imai, Ronald L. Rivest, and Tsutomu Matsumoto, editors, *ASIACRYPT'91*, volume 739 of *LNCS*, pages 210–224. Springer, Heidelberg, November 1993. doi:10.1007/3-540-57332-1_17.

[EM97] Shimon Even and Yishay Mansour. A construction of a cipher from a single pseudorandom permutation. *Journal of Cryptology*, 10(3):151–162, June 1997. doi:10.1007/s001459900025.

[Eve88] Jan-Hendrik Evertse. Linear structures in blockciphers. In David Chaum and Wyn L. Price, editors, *EUROCRYPT'87*, volume 304 of *LNCS*, pages 249–266. Springer, Heidelberg, April 1988. doi:10.1007/3-540-39118-5_23.

[Gho22] Santosh Ghosh. Low-latency crypto: An emerging paradigm of lightweight cryptography. Presented in Lightweight Cryptography Workshop 2022, 2022. URL: https://csrc.nist.gov/Presentations/2022/low-latency-crypto-an-emerging-paradigm-of-lightwe.

[GJN+16] Jian Guo, Jérémy Jean, Ivica Nikolic, Kexin Qiao, Yu Sasaki, and Siang Meng Sim. Invariant subspace attack against Midori64 and the resistance criteria for S-box designs. *IACR Trans. Symm. Cryptol.*, 2016(1):33–56, 2016. https://tosc.iacr.org/index.php/ToSC/article/view/534. doi:10.13154/tosc.v2016.i1.33-56.

[Gue16] Shay Gueron. A memory encryption engine suitable for general purpose processors. Cryptology ePrint Archive, Paper 2016/204, 2016. https://eprint.iacr.org/2016/204. URL: https://eprint.iacr.org/2016/204.

[HSH+09] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. Lest we remember: cold-boot attacks on encryption keys. *Commun. ACM*, 52(5):91–98, 2009. doi:10.1145/1506409.1506429.

[HWKS98] Chris Hall, David Wagner, John Kelsey, and Bruce Schneier. Building PRFs from PRPs. In Hugo Krawczyk, editor, *CRYPTO'98*, volume 1462 of *LNCS*, pages 370–389. Springer, Heidelberg, August 1998. doi:10.1007/BFb0055742.

[IMO+22] Akiko Inoue, Kazuhiko Minematsu, Maya Oda, Rei Ueno, and Naofumi Homma. ELM: A Low-Latency and Scalable Memory Encryption Scheme. *IEEE Transactions on Information Forensics and Security*, 17:2628–2643, 2022. doi:10.1109/TIFS.2022.3188146.

[Int20] Intel. Intel trust domain extensions. Whitepaper, 2020.

[KDGD20] Michael Kounavis, Sergej Deutsch, Santosh Ghosh, and David Durham. K-cipher: A low latency, bit length parameterizable cipher. Cryptology ePrint Archive, Paper 2020/030, 2020. https://eprint.iacr.org/2020/030. URL: https://eprint.iacr.org/2020/030.

[KHSH20] Seonggyeom Kim, Deukjo Hong, Jaechul Sung, and Seokhie Hong. Classification of 4-bit s-boxes for bogi permutation. *IEEE Access*, 8:210935–210949, 2020. doi:10.1109/ACCESS.2020.3039273.

[Knu95] Lars R. Knudsen. Truncated and higher order differentials. In Bart Preneel, editor, *FSE'94*, volume 1008 of *LNCS*, pages 196–211. Springer, Heidelberg, December 1995. doi:10.1007/3-540-60590-8_16.

[LAAZ11] Gregor Leander, Mohamed Ahmed Abdelraheem, Hoda AlKhzaimi, and Erik Zenner. A cryptanalysis of PRINTcipher: The invariant subspace attack. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 206–221. Springer, Heidelberg, August 2011. doi:10.1007/978-3-642-22792-9_12.

[Lai95] Xuejia Lai. Additive and linear structures of cryptographic functions. In Bart Preneel, editor, *FSE'94*, volume 1008 of *LNCS*, pages 75–85. Springer, Heidelberg, December 1995. doi:10.1007/3-540-60590-8_6.

[LMMR21] Gregor Leander, Thorben Moos, Amir Moradi, and Shahram Rasoolzadeh. The SPEEDY family of block ciphers engineering an ultra low-latency cipher from gate level for secure processor architectures. *IACR TCHES*, 2021(4):510–545, 2021. https://tches.iacr.org/index.php/TCHES/article/view/9074. doi:10.46586/tches.v2021.i4.510-545.

[LMR15] Gregor Leander, Brice Minaud, and Sondre Rønjom. A generic approach to invariant subspace attacks: Cryptanalysis of robin, iSCREAM and Zorro. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 254–283. Springer, Heidelberg, April 2015. doi:10.1007/978-3-662-46800-5_11.

[LRD+21] Michael LeMay, Joydeep Rakshit, Sergej Deutsch, David M. Durham, Santosh Ghosh, Anant Nori, Jayesh Gaur, Andrew Weiler, Salmin Sultana, Karanvir Grewal, and Sreenivas Subramoney. Cryptographic capability computing. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '21, page 253–267, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3466752.3480076.

[LWW+22] Mengyuan Li, Luca Wilke, Jan Wichelmann, Thomas Eisenbarth, Radu Teodorescu, and Yinqian Zhang. A systematic look at ciphertext side channels on AMD SEV-SNP. In *43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022*, pages 337–351. IEEE, 2022. doi:10.1109/SP46214.2022.9833768.

[MDA17] Silvia Mella, Joan Daemen, and Gilles Van Assche. New techniques for trail bounds and application to differential trails in Keccak. *IACR Trans. Symm. Cryptol.*, 2017(1):329–357, 2017. doi:10.13154/tosc.v2017.i1.329-357.

[MKPA22] Mohammad Mahzoun, Liliya Kraleva, Raluca Posteuca, and Tomer Ashur. Differential cryptanalysis of K-cipher. In *IEEE Symposium on Computers and Communications, ISCC 2022, Rhodes, Greece, June 30 - July 3, 2022*, pages 1–7. IEEE, 2022. doi:10.1109/ISCC55528.2022.9912926.

[MMGD22] Alireza Mehrdad, Silvia Mella, Lorenzo Grassi, and Joan Daemen. Differential trail search in cryptographic primitives with big-circle chi:: Application to subterranean. *IACR Transactions on Symmetric Cryptology*, 2022(2):253–288, Jun. 2022. URL: https://tosc.iacr.org/index.php/ToSC/article/view/9721, doi:10.46586/tosc.v2022.i2.253-288.

[MP13] Nicky Mouha and Bart Preneel. Towards finding optimal differential characteristics for ARX: Application to Salsa20. Cryptology ePrint Archive, Report 2013/328, 2013. https://eprint.iacr.org/2013/328.

[MWGP12] Nicky Mouha, Qingju Wang, Dawu Gu, and Bart Preneel. Differential and linear cryptanalysis using mixed-integer linear programming. In Chuan-Kun Wu, Moti Yung, and Dongdai Lin, editors, *Information Security and Cryptology*, pages 57–76, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. doi:10.1007/978-3-642-34704-7_5.

[SHW+14] Siwei Sun, Lei Hu, Peng Wang, Kexin Qiao, Xiaoshuang Ma, and Ling Song. Automatic security evaluation and (related-key) differential characteristic search: Application to SIMON, PRESENT, LBlock, DES(L) and other bit-oriented block ciphers. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part I*, volume 8873 of *LNCS*, pages 158–178. Springer, Heidelberg, December 2014. doi:10.1007/978-3-662-45611-8_9.

[SWW17] Ling Sun, Wei Wang, and Meiqin Wang. Automatic search of bit-based division property for ARX ciphers and word-based division property. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*, pages 128–157. Springer, Heidelberg, December 2017. doi:10.1007/978-3-319-70694-8_5.

[SWW21] Ling Sun, Wei Wang, and Meiqin Wang. Accelerating the search of differential and linear characteristics with the SAT method. *IACR Trans. Symm. Cryptol.*, 2021(1):269–315, 2021. doi:10.46586/tosc.v2021.i1.269-315.

[TLS16] Yosuke Todo, Gregor Leander, and Yu Sasaki. Nonlinear invariant attack - practical attack on full SCREAM, iSCREAM, and Midori64. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 3–33. Springer, Heidelberg, December 2016. doi:10.1007/978-3-662-53890-6_1.

[TM16] Yosuke Todo and Masakatu Morii. Bit-based division property and application to simon family. In Thomas Peyrin, editor, *FSE 2016*, volume 9783 of *LNCS*, pages 357–377. Springer, Heidelberg, March 2016. doi:10.1007/978-3-662-52993-5_18.

[Tod15] Yosuke Todo. Structural evaluation by generalized integral property. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 287–314. Springer, Heidelberg, April 2015. doi:10.1007/978-3-662-46800-5_12.

[WC81] Mark N. Wegman and J.Lawrence Carter. New hash functions and their use in authentication and set equality. *Journal of Computer and System Sciences*, 22(3):265–279, 1981. URL: https://www.sciencedirect.com/science/article/pii/0022000081900337, doi:10.1016/0022-0000(81)90033-7.

[WCJ+21] Yoo-Seung Won, Soham Chatterjee, Dirmanto Jap, Arindam Basu, and Shivam Bhasin. Deepfreeze: Cold boot attacks and high fidelity model recovery on commercial edgeml device. In *IEEE/ACM International Conference On Computer Aided Design, ICCAD 2021, Munich, Germany, November 1-4, 2021*, pages 1–9. IEEE, 2021. doi:10.1109/ICCAD51958.2021.9643512.

[XZBL16] Zejun Xiang, Wentao Zhang, Zhenzhen Bao, and Dongdai Lin. Applying MILP method to searching integral distinguishers based on division property for 6 lightweight block ciphers. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 648–678. Springer, Heidelberg, December 2016. doi:10.1007/978-3-662-53887-6_24.

[YADA17] Salessawi Ferede Yitbarek, Misiker Tadesse Aga, Reetuparna Das, and Todd M. Austin. Cold boot attacks are still hot: Security analysis of memory scramblers in modern processors. In *2017 IEEE International Symposium on High Performance Computer Architecture, HPCA 2017, Austin, TX, USA, February 4-8, 2017*, pages 313–324. IEEE Computer Society, 2017. doi:10.1109/HPCA.2017.10.

[Yal22] Tolga Yalcin. Need for low-latency ciphers - a comparative study of nist lwc finalists. Presented in Lightweight Cryptography Workshop 2022, 2022. URL: https://csrc.nist.gov/Presentations/2022/need-for-low-latency-ciphers-a-comparative-study-o.

[ZBL+14] Wentao Zhang, Zhenzhen Bao, Dongdai Lin, Vincent Rijmen, Bohan Yang, and Ingrid Verbauwhede. RECTANGLE: A bit-slice lightweight block cipher suitable for multiple platforms. Cryptology ePrint Archive, Report 2014/084, 2014. https://eprint.iacr.org/2014/084.

A Latency Comparison of Different 4-bit S-boxes

Table 13: Latency comparison of different 4-bit S-boxes

Cipher	Delay (ns)
Midori Sb0	0.26
Midori Sb1	0.22
Orthors	0.22
QARMA σ_0	0.23
QARMA σ_1	0.23
QARMA σ_2	0.25
QARMA σ_2^{-1}	0.26
PRINCE	0.24
B0	0.26
B1	0.26
B2	0.27
B3	0.31
B4	0.3
B5	0.27

B6	0.3
B7	0.28
B8	0.28
B9	0.26
B10	0.26
B11	0.26
B12	0.31
B13	0.25
B14	0.27
B15	0.27
B16	0.26
(0, 4)-Num1-DL-0	0.28
(0, 4)-Num1-DL-1	0.31
(1, 3)-Num1-DL-0	0.25
(1, 3)-Num1-DL-1	0.29
(1, 3)-Num1-DL-2	0.27
(1, 3)-Num1-DL-3	0.28
(2, 2)-Num1-DL-0	0.27
(2, 2)-Num1-DL-1	0.27
(2, 2)-Num1-DL-2	0.3
(2, 2)-Num1-DL-3	0.26

B The Specification of RC_i

$RC_1 =$ 0x243f6a8885a308d313198a2e03707344a4093822299f31d0082efa98ec4e6c89
452821e638d01377be5466cf34e90c6cc0ac29b7c97c50dd3f84d5b5b5470917
9216d5d98979fb1bd1310ba698dfb5ac2ffd72dbd01adfb7b8e1afed6a267e96
ba7c9045f12c7f9924a19947b3916cf70801f2e2858efc16636920d871574e69
a458fea3f4933d7e0d95748f728eb658718bcd5882154aee7b54a41dc25a59b5.

$RC_2 =$ 0x9c30d5392af26013c5d1b023286085f0ca417918b8db38ef8e79dcb0603a180e
6c9e0e8bb01e8a3ed71577c1bd314b2778af2fda55605c60e65525f3aa55ab94
5748986263e8144055ca396a2aab10b6b4cc5c341141e8cea15486af7c72e993
b3ee1411636fbc2a2ba9c55d741831f6ce5c3e169b87931eafd6ba336c24cf5c
7a325381289586773b8f48986b4bb9afc4bfe81b6628219361d809ccfb21a991.

$RC_3 =$ 0x487cac605dec8032ef845d5de98575b1dc262302eb651b8823893e81d396acc5
0f6d6ff383f442392e0b4482a484200469c8f04a9e1f9b5e21c66842f6e96c9a
670c9c61abd388f06a51a0d2d8542f68960fa728ab5133a36eef0b6c137a3be4
ba3bf0507efb2a98a1f1651d39af017666ca593e82430e888cee8619456f9fb4
7d84a5c33b8b5ebee06f75d885c12073401a449f56c16aa64ed3aa62363f7706.

$RC_4 =$ 0x1bfedf72429b023d37d0d724d00a1248db0fead349f1c09b075372c980991b7b
25d479d8f6e8def7e3fe501ab6794c3b976ce0bd04c006bac1a94fb6409f60c4
5e5c9ec2196a246368fb6faf3e6c53b51339b2eb3b52ec6f6dfc511f9b30952c
cc814544af5ebd09bee3d004de334afd660f2807192e4bb3c0cba85745c8740f
d20b5f39b9d3fbdb5579c0bd1a60320ad6a100c6402c7279679f25fefb1fa3cc.

$RC_5 =$ 0x8ea5e9f8db3222f83c7516dffd616b152f501ec8ad0552ab323db5fafd238760
53317b483e00df829e5c57bbca6f8ca01a87562edf1769dbd542a8f6287effc3
ac6732c68c4f5573695b27b0bbca58c8e1ffa35db8f011a010fa3d98fd2183b8
4afcb56c2dd1d35b9a53e479b6f84565d28e49bc4bfb9790e1ddf2daa4cb7e33
62fb1341cee4c6e8ef20cada36774c01d07e9efe2bf11fb495dbda4dae909198.

$RC_6 =$ 0xeaad8e716b93d5a0d08ed1d0afc725e08e3c5b2f8e7594b78ff6e2fbf2122b64
8888b812900df01c4fad5ea0688fc31cd1cff191b3a8c1ad2f2f2218be0e1777
ea752dfe8b021fa1e5a0cc0fb56f74e818acf3d6ce89e299b4a84fe0fd13e0b7
7cc43b81d2ada8d9165fa2668095770593cc7314211a1477e6ad206577b5fa86
c75442f5fb9d35cfebcdaf0c7b3e89a0d6411bd3ae1e7e4900250e2d2071b35e.

$RC_7 =$ 0x226800bb57b8e0af2464369bf009b91e5563911d59dfa6aa78c14389d95a537f
207d5ba202e5b9c5832603766295cfa911c819684e734a41b3472dca7b14a94a
1b5100529a532915d60f573fbc9bc6e42b60a47681e6740008ba6fb5571be91f
f296ec6b2a0dd915b6636521e7b9f9b6ff34052ec585566453b02d5da99f8fa1
08ba47996e85076a4b7a70e9b5b32944db75092ec4192623ad6ea6b049a7df7d.

$RC_8 =$ 0x9cee60b88fedb266ecaa8c71699a17ff5664526cc2b19ee1193602a575094c29
a0591340e4183a3e3f54989a5b429d656b8fe4d699f73fd6a1d29c07efe830f5
4d2d38e6f0255dc14cdd20868470eb266382e9c6021ecc5e09686b3f3ebaefc9
3c9718146b6a70a1687f358452a0e286b79c5305aa5007373e07841c7fdeae5c
8e7d44ec5716f2b8b03ada37f0500c0df01c1f040200b3ffae0cf51a3cb574b2.

$RC_9 =$ 0x25837a58dc0921bdd19113f97ca92ff69432477322f547013ae5e58137c2dadc
c8b576349af3dda7a94461460fd0030eecc8c73ea4751e41e238cd993bea0e2f
3280bba1183eb3314e548b384f6db9086f420d03f60a04bf2cb8129024977c79
5679b072bcaf89afde9a771fd9930810b38bae12dccf3f2e5512721f2e6b7124
501adde69f84cd877a5847187408da17bc9f9abce94b7d8cec7aec3adb851dfa.

$RC_{10} =$ 0x63094366c464c3d2ef1c18473215d908dd433b3724c2ba1612a14d432a65c451
50940002133ae4dd71dff89e10314e5581ac77d65f11199b043556f1d7a3c76b
3c11183b5924a509f28fe6ed97f1fbfa9ebabf2c1e153c6e86e34570eae96fb1
860e5e0a5a3e2ab3771fe71c4e3d06fa2965dcb999e71d0f803e89d65266c825
2e4cc9789c10b36ac6150eba94e2ea78a5fc3c531e0a2df4f2f74ea7361d2b3d.

$RC_{11} =$ 0x1939260f19c279605223a708f71312b6ebadfe6eeac31f66e3bc4595a67bc883
b17f37d1018cff28c332ddefbe6c5aa56558218568ab9802eecea50fdb2f953b
2aef7dad5b6e2f841521b62829076170ecdd4775619f151013cca830eb61bd96
0334fe1eaa0363cfb5735c904c70a239d59e9e0bcbaade14eecc86bc60622ca7
9cab5cabb2f3846e648b1eaf19bdf0caa02369b9655abb5040685a323c2ab4b3.

$$
\begin{aligned}
RC_{12} = {} & \texttt{0x319ee9d5c021b8f79b540b19875fa09995f7997e623d7da8f837889a97e32d77} \\
& \texttt{11ed935f166812810e358829c7e61fd696dedfa17858ba9957f584a51b227263} \\
& \texttt{9b83c3ff1ac24696cdb30aeb532e30548fd948e46dbc312858ebf2ef34c6ffea} \\
& \texttt{fe28ed61ee7c3c735d4a14d9e864b7e342105d14203e13e045eee2b6a3aaabea} \\
& \texttt{db6c4f15facb4fd0c742f442ef6abbb5654f3b1d41cd2105d81e799e86854dc7}.
\end{aligned}
$$

$$
\begin{aligned}
RC_{13} = {} & \texttt{0xe44b476a3d816250cf62a1f25b8d2646fc8883a0c1c7b6a37f1524c369cb7492} \\
& \texttt{47848a0b5692b285095bbf00ad19489d1462b17423820e0058428d2a0c55f5ea} \\
& \texttt{1dadf43e233f70613372f0928d937e41d65fecf16c223bdb7cde3759cbee7460} \\
& \texttt{4085f2a7ce77326ea607808419f8509ee8efd85561d99735a969a7aac50c06c2} \\
& \texttt{5a04abfc800bcadc9e447a2ec3453484fdd567050e1e9ec9db73dbd3105588cd}.
\end{aligned}
$$

$$
\begin{aligned}
RC_{14} = {} & \texttt{0x675fda79e3674340c5c43465713e38d83d28f89ef16dff20153e21e78fb03d4a} \\
& \texttt{e6e39f2bdb83adf7e93d5a68948140f7f64c261c94692934411520f77602d4f7} \\
& \texttt{bcf46b2ed4a20068d40824713320f46a43b7d4b7500061af1e39f62e97244546} \\
& \texttt{14214f74bf8b88404d95fc1d96b591af70f4ddd366a02f45bfbc09ec03bd9785} \\
& \texttt{7fac6dd031cb850496eb27b355fd3941da2547e6abca0a9a28507825530429f4}.
\end{aligned}
$$

$$
\begin{aligned}
RC_{15} = {} & \texttt{0x0a2c86dae9b66dfb68dc1462d7486900680ec0a427a18dee4f3ffea2e887ad8c} \\
& \texttt{b58ce0067af4d6b6aace1e7cd3375fecce78a399406b2a4220fe9e35d9f385b9} \\
& \texttt{ee39d7ab3b124e8b1dc9faf74b6d185626a36631eae397b23a6efa74dd5b4332} \\
& \texttt{6841e7f7ca7820fbfb0af54ed8feb397454056acba48952755533a3a20838d87} \\
& \texttt{fe6ba9b7d096954b55a867bca1159a58cca9296399e1db33a62a4a563f3125f9}.
\end{aligned}
$$

$$
\begin{aligned}
RC_{16} = {} & \texttt{0x5ef47e1c9029317cfdf8e80204272f7080bb155c05282ce395c11548e4c66d22} \\
& \texttt{48c1133fc70f86dc07f9c9ee41041f0f404779a45d886e17325f51ebd59bc0d1} \\
& \texttt{f2bcc18f41113564257b7834602a9c60dff8e8a31f636c1b0e12b4c202e1329e} \\
& \texttt{af664fd1cad181156b2395e0333e92e13b240b62eebeb92285b2a20ee6ba0d99} \\
& \texttt{de720c8c2da2f728d012784595b794fd647d0862e7ccf5f05449a36f877d48fa}.
\end{aligned}
$$

$$
\begin{aligned}
RC_{17} = {} & \texttt{0xc39dfd27f33e8d1e0a476341992eff743a6f6eabf4f8fd37a812dc60a1ebddf8} \\
& \texttt{991be14cdb6e6b0dc67b55106d672c372765d43bdcd0e804f1290dc7cc00ffa3} \\
& \texttt{b5390f92690fed0b667b9ffbcedb7d9ca091cf0bd9155ea3bb132f88515bad24} \\
& \texttt{7b9479bf763bd6eb37392eb3cc1159798026e297f42e312d6842ada7c66a2b3b} \\
& \texttt{12754ccc782ef11c6a124237b79251e706a1bbe64bfb63501a6b101811caedfa}.
\end{aligned}
$$

$$
\begin{aligned}
RC_{18} = {} & \texttt{0x3d25bdd8e2e1c3c9444216590a121386d90cec6ed5abea2a64af674eda86a85f} \\
& \texttt{bebfe98864e4c3fe9dbc8057f0f7c08660787bf86003604dd1fd8346f6381fb0} \\
& \texttt{7745ae04d736fccc83426b33f01eab71b08041873c005e5f77a057bebde8ae24} \\
& \texttt{55464299bf582e614e58f48ff2ddfda2f474ef388789bdc25366f9c3c8b38e74} \\
& \texttt{b475f25546fcd9b97aeb26618b1ddf84846a0e79915f95e2466e598e20b45770}.
\end{aligned}
$$

C The Critical Path in AE/VD Process with Nonce Pre-Generated

In cases where there is a significant gap between the availability of the nonce and the plaintext, the keystream and mask may be generated before the plaintext arrives. The critical path of this scenario is illustrated in Figure 5. The latency of the AE process in this case is limited to a UHF delay plus an XOR delay. The VD process, on the other hand, incurs an additional XOR delay compared to the AE process. The UHF delay is deterministic for both the AE and VD processes. However, the latency of `Twinkle` is still crucial as it needs to be less than the latency gap between the availability of the nonce and the plaintext for this case to occur.

Pre-computation acceleration is not applicable for tweakable block ciphers or operation modes that involve inputting plaintext into a block cipher. Pre-computation may not always be an option, so it was not included in the hardware evaluation. But it is crucial to factor it into the design considerations.

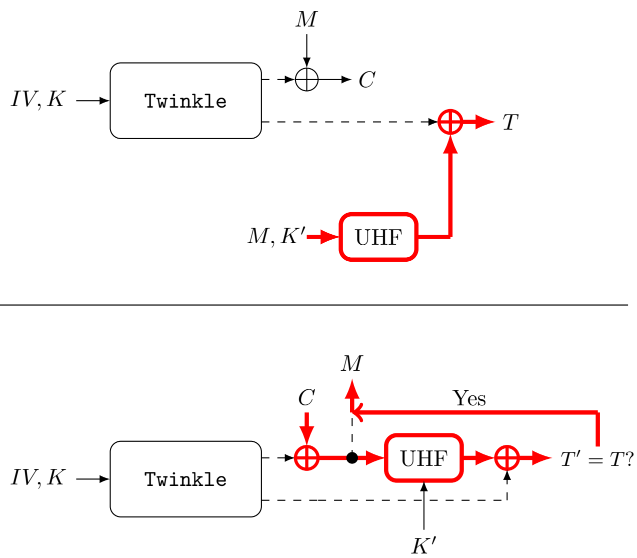

Figure 5: The critical path in encryption and authentication process(top) and decryption and verification process(bottom) with Nonce pre-generated

D Results of Security Evaluation

D.1 Differential Trail

Refer to Table 14.

D.2 Linear Trail

Refer to Table 15.

D.3 Truncated Difference Trails

Refer to Table 16 and Table 17.

Table 14: The differential trail of `Twinkle` with 28 active S-boxes

Operations	Output difference
S-box	....................8...4..1.. ....1.........................4............. ....2...1.........................8... ....1.................
$\mathcal{R}$	....1..................................
S-box $\circ\,\mathcal{R}$	....8..................................
$\mathcal{R}^2$	....4................ ....8......................... ....8................................
S-box $\circ\,\mathcal{R}^2$	....2................ ....3......................... ....9................................
$\mathcal{R}^3$	.....................8...4.....2........ ....1.........2....8.....1.. ....1...2..............1.......8....................... ....2...................1..................8...1..........................

Table 16: The truncated differential trail of `Twinkle`^{128}/`Twinkle`^{256} with a probability of 1

	Last 3.5-round truncated differential trail
Δ_{in}	10000000000000000000000000000000000000000000000000000000000000000000000000000000 10000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000 10000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000
Round 1	00000000000000000000000000000000000000000000000000000000000000000000000000000?00 00000000000?00000000000000000000000000000000000000000000000000000000000000000000 00000000?00000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000?00000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000?000 00000000000000000000000000000000000000000000000000000000?00000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000000000000000000? 00000000000000000000000000000000000000000000?00000000000000000000000000000000000 000000000000000000000000000?0000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000000000000000000000000000
Round 2	00000?00?000000000000000?0000000000000000000?00000000000000000000?00000000?00000 00000000?00000000?0?00?00?0000000000000000000?0000000000000000?000?0000000000000 00000?000000?000?00?0000000?00000?000000000000000000000?0000000000000000?0000000 0000000?0000000000?00000?0000000000000000?0000000000?00?0?0?00000000000000000?00 0000000000000?00?0000000000000?000000000000000000000000?000000000?000000000?0000 00000000000000?0000000000?0??000000000000000000?00000000000000000000?00000000000 0000?00?00000000000000000000000000000000000000?0000000000000000000?000000?000000 000000000000000000000000000000?0000000000000000000?00?000000000?00?000?000000000

	00000000?0000000000000000?000000000??00?000000000000?000000000000000000000000 ??00000000000000000000?00000000000?00000000000000000000000?0?000000000000000 00000000?000000?00000000000000000?0000000?00000000000?0000000000000?00000000 000000000??000000000000000?0000?000000000000000000000000000?00000000?00000000 0?00000?00?0000000000000000?00000?0000000000?00000000?0000000000000000000?000 00000000000000?0000000000000000000000000?0000000?00?00?0000000000000000000?00 000?0000000000000000000?000000?000?00?000000000?000000000000000000000000000 0000000000?000000000000000?0000000000000000?0000000000?00?00?0000000000000000
Round 3	00????0?0??00????0???????000?0?0????????0???000??00?0????0?0????0000?0?0????0?0 ?00?0?00?00000???0??0?????0????0???????0?0?0??0?0?00??0??00?00???????0?00??0??0? ?????0?0???0?0??000?00?????0?????0?00?00????0?00?00??0?0?0??0??0???0???0??0??00 0000??0???0?00??????0?0?0??0?????00??0??0??0000???0??0?0??00???0????????0?0??00 ?00?00?????0??0?0?000?00??????0?????00?00???0??0??0?0??00???00000????????0??0???0?0?0 0???0?000?0????00??00??????00???????????0??0?0?0?0?00?0???0??0?0??0?0?00?0???0?? ??0????0????????0??0??0?000?0??0000???00??0???00?00?0??????00?????00??00??0??00 ?????00?0?0000000?00??00??0???0?0???00?????0??0?????0???000??0?0??0???0?0????00?? 0?0????0?0???????00???0?00?000??0??0??????00?????0?????????00??0?????0??0??00000 ???0??00??0??00?00?????000000??0??0?0?0??00?00???????0??0???0?00?0??0?0?0?????????? 0??00??00??0?????00??0??????00?0?0??0000????0???0???????00???000?0????0?0???000? ????????0?0?0?00???0??0?0?0?????00?00???0??0?00?00?0000?0?????0???0?0?00?0???0?? 000?????0?0??00????00?0????00?0????00?00?00??????0???0?0???00?0????0000???0??0?? ???00?0???0?00???0??0???0??????0?00??0?00?000?????0???0???00?????0??0?????00?0? ???0????0??????000??0?0??0?0?00?????000???????00??????0?0???00?0??0?00??000?00? ??00000????0???000?0??0???0??????????0???0?00?000000??0??00??0?0????0??0??0?0?0?
Δ_{out}	?????????????????????????????????0?? ??????????????????????????0??? ????????????0?? ??0????????????????? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??

Table 17: The truncated differential trail of `Twinkle`^{64} with a probability of $2^{-7.4}$

	Last 4-round truncated differential trail
Δ_{in}	0000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000 1000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000 1000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000
Sb $p = \frac{3}{8}$	0000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000 ?000000000000000000000000000000000000000000000000000000000000000 1000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000

	00000000000000000000000000000000000000000000000000000000000000000000000000000000
$\mathbf{S} \circ \mathcal{R}$ $p = 2^{-5}$	0?00000000000000?000000000000000000000000000000000000000000000000000000000000000
	0?00000000000000?000000000000000000000000000000000000000000000000000000000000000
	0000000000000000?000000000000000000000000000000000000000000000000000000000000000
	0000000000000000?000000000000000000000000000000000000000000000000000000000000000
	000000000000000000000000000000000000000000000000000000?000000000000000?000000000
	000000000000000000000000000000000000000000000000000000?000000000000000?000000000
	000000000000000000000000000000000000000000000000000000?000000000000000?000000000
	000000000000000000000000000000000000000000000000000000?000000000000000?000000000
	00000000000000?00000000000000000000000000000000000000000000000000000000000000000
	00000000000000?00000000000000000000000000000000000000000000000000000000000000000
	00000000000000?00000000000000000000000000000000000000000000000000000000000000000
	00000000000000?00000000000000000000000000000000000000000000000000000000000000000
	00000000000000000000000000000000000000000000000000000000000100000000000000000000
	00000000000000000000000000000000000000000000000000000000000000000000000000000000
	00000000000000000000000000000000000000000000000000000000000000000000000000000000
	00000000000000000000000000000000000000000000000000000000000100000000000000000000
$\mathbf{S} \circ \mathcal{R}^2$ $p = 2^{-1}$	000000?00000??00000000???00?0000000000??000000000000000?0000??00000000??00000??0
	000000?00000??00000000???00?0000000000??000000000000000?0000??00000000??00000??0
	000000?00000??00000000???00?0000000000??000000000000000?0000??00000000??00000??0
	000000?00000??00000000???00?0000000000??000000000000000?0000??00000000??00000??0
	000000000000?00000?000000?00?0000?00?0000?00??0000000?0000?0?00000?0??00?0000?00
	000000000000?00000?000000?00?0000?00?0000?00??0000000?0000?0?00000?0??00?0000?00
	000000000000?00000?000000?00?0000?00?0000?00??0000000?0000?0?00000?0??00?0000?00
	000000000000?00000?000000?00?0000?00?0000?00?10000000?0000?0?00000?0??00?0000?00
	0000?000?0000?000000000000000?00000000000000??0000000?00000?00000000?0000?0?0000
	0000?000?0000?000000000000000?00000000000?00??0000000?00000?00000000?0000?0?0000
	0000?000?0000?000000000000000?00000000000?00??0000000?00000?00000000?0000?0?0000
	0000?000?0000?000000000000000?00000000000?00??0000000?00000?00000000?0000?0?0000
	?00000?000000?0?00000000000000000?00000000??0?000?0000000000??00?000000000000000
	?00000?000000?0?00000000000000000?00000000??0?000?0000000000??00?000000000000000
	?00000?000000?0?00000000000000000?00000000??0?000?0000000000??00?000000000000000
	?00000?000000?0?00000000000000000?00000000??0?000?0000000000?100?000000000000000
$\mathbf{S} \circ \mathcal{R}^3$ $p = 1$	0??????0???????00??????????????????????0????????0????0????????0?00??????????????
	0??????0???????00??????????????????????0????????0????0????????0?00??????????????
	0??????0???????00??????????????????????0????????0????0????????0?00??????????????
	0??????0???????00??????????????????????0????????0????0????????0?00??????????????
	??????0??????0???????00???????????????0??????????????????????0??0?????0?????????
	??????0??????0???????00???????????????0??????????????????????0??0?????0?????????
	??????0??????0???????00???????????????0??????????????????????0??0?????0?????????
	??????0??????0???????00???????????????0??????????????????????0??0?????0?????????
	??????????????0??????0???????0???????0??0????0??0???????????00????0?????????0???
	??????????????0??????0???????0???????0??0????0??0???????????00????0?????????0???
	??????????????0??????0???????0???????0??0????0??0???????????00????0?????????0???
	??????????????0??????0???????0???????0??0????0??0???????????00????0?????????0???
	?0???????0?????????????????0???????????????????????????0?????????????????0??????
	?0???????0?????????????????0???????????????????????????0?????????????????0??????
	?0???????0?????????????????0???????????????????????????0?????????????????0??????
	?0???????0?????????????????0???????????????????????????0?????????????????0??????
Δ_{out}	????
	????
	????
	????
	????
	????
	???0
	????
	????
	????
	????
	????
	????
	????
	????
	????

Table 15: The linear trail of `Twinkle` with 28 active S-boxes

Operations	Output mask
S-box	.........2........8..............................1......1....................... 8..........1..........................1... ....2..1........................4........8........................ ....8..................1..............4..........................8...
$\mathcal{R}$	....1.............. .....c.. ....9..
S-box $\circ\,\mathcal{R}$	....2.............. .....1.. ....1..
$\mathcal{R}^2$	....1 ....
S-box $\circ\,\mathcal{R}^2$	....2 ....
$\mathcal{R}^3$	....4..........2 .8...2.......................... ....4..28.................... ....2...1.

E Test Vectors

E.1 `Twinkle`^{64}

K = 0x0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000

IV = 0x00000000000000000000000000000000

O = 0x94e02b57b26533584379d913a624479abaa5350226bbbd54c06a68e79f544a8a
87c9637257d94c75a6d80e18fd56157d7e00dcca6366691c307ff99695ed7063
046c8b7940641c8be53c562d591fb71abe75c8d6d474b2806f373b49c55248ae
f4d84ef3336fac360d26487ff234776bcb7f24c1531884bb7b16195d343429c3
a5f3f19df1dce04c76ddf79f402ca5cc

E.2 **Twinkle**^{128}

$K =$ `0x0000000000000000000000000000000000000000000000000000000000000000`
`0000000000000000000000000000000000000000000000000000000000000000`
`0000000000000000000000000000000000000000000000000000000000000000`
`0000000000000000000000000000000000000000000000000000000000000000`

$IV =$ `0x00000000000000000000000000000000`

$O =$ `0xdb177b7356bd5bbac97b11626fc1fdcf20e3b1dc2e47677161762e9d7ce5a980`
`b02e7bfe101e689d3a44a3378edb1bc1edf86800a48c7fc49e75b45b05f98456`
`0d663630cc66313b4b65b56de7ebfd671411fb366fdcfd0d32650fc9a6c10e9b`
`cb176f97f92a652c192ba8cd7590fe39fd18e3cc855360da1cef031c8222b6f6`
`bf0b8e4504d37b8d64fa78eb8eed9a7d`

E.3 **Twinkle**^{256}

$K =$ `0x0000000000000000000000000000000000000000000000000000000000000000`
`0000000000000000000000000000000000000000000000000000000000000000`
`0000000000000000000000000000000000000000000000000000000000000000`
`0000000000000000000000000000000000000000000000000000000000000000`

$IV =$ `0x00000000000000000000000000000000`

$O =$ `0x181d3de8a1c97b9fb38a06a182ae7472763eb6b8523b2b11088da5941c09087a`
`d4d3d45b45e4d9f27ee7ef865711694548d4f22cdddcd98e4a5d27b80a47d31e`
`dcadfd08333bcf49b15389d6f438c5c32fc3fa950c91e7a8cc1127bfcb2d73a4`
`58c196e5ee71166b4a7fa80048621cd5635c7997296a8df8e26548c93b9732fb`
`35800f0e727a6ddf65db82b704e0d46f`