IACR Transactions on Symmetric Cryptology
ISSN 2519-173X, Vol. 2024, No. 1, pp. 158–187. DOI:10.46586/tosc.v2024.i1.158-187

Improved Meet-in-the-Middle Nostradamus Attacks on AES-like Hashing

Xiaoyang Dong[1,3,5,6], Jian Guo[2], Shun Li[4,2(✉)], Phuong Pham[2] and Tianyu Zhang[2]

[1] Institute for Network Sciences and Cyberspace, BNRist, Tsinghua University, Beijing, China
xiaoyangdong@tsinghua.edu.cn
[2] Nanyang Technological University, Singapore, Singapore
guojian@ntu.edu.sg,{pham0079,tianyu005}@e.ntu.edu.sg
[3] State Key Laboratory of Cryptology, P.O.Box 5159, Beijing, 100878, China
[4] School of Cryptology, University of Chinese Academy of Sciences, Beijing, China
lishun@ucas.ac.cn
[5] Zhongguancun Laboratory, Beijing, China
[6] Shandong Institute of Blockchain, Jinan, China

Abstract. The Nostradamus attack was originally proposed as a security vulnerability for a hash function by Kelsey and Kohno at EUROCRYPT 2006. It requires the attacker to commit to a hash value y of an iterated hash function H. Subsequently, upon being provided with a message prefix P, the adversary's task is to identify a suffix S such that $H(P\|S)$ equals y. Kelsey and Kohno demonstrated a herding attack requiring $O(\sqrt{n} \cdot 2^{2n/3})$ evaluations of the compression function of H, where n represents the output and state size of the hash, placing this attack between preimage attacks and collision searches in terms of complexity. At ASIACRYPT 2022, Benedikt *et al.* transform Kelsey and Kohno's attack into a quantum variant, decreasing the time complexity from $O(\sqrt{n} \cdot 2^{2n/3})$ to $O(\sqrt[3]{n} \cdot 2^{3n/7})$. At ToSC 2023, Zhang *et al.* proposed the first dedicated Nostradamus attack on AES-like hashing in both classical and quantum settings. In this paper, we have made revisions to the multi-target technique incorporated into the meet-in-the-middle automatic search framework. This modification leads to a decrease in time complexity during the online linking phase, effectively reducing the overall attack time complexity in both classical and quantum scenarios. Specifically, we can achieve more rounds in the classical setting and reduce the time complexity for the same round in the quantum setting.

Keywords: Hash Function · Meet-in-the-middle Attack · AES-like · Nostradamus Attack

1 Introduction

Cryptographic hash functions serve as fundamental cryptographic primitives, forming the basis for numerous advanced cryptographic protocols like digital signatures, authenticated encryption, secure multiparty computation, post-quantum public-key cryptography, and more. To be considered secure, a cryptographic hash function must satisfy three essential security properties: collision resistance, preimage resistance, and second-preimage resistance.

AES-like hashing design utilizes block ciphers or permutations with features similar to AES, such as Whirlpool [BR+00], Grøstl [GKM+09], AES-MMO, Grindahl [KRT07], ECHO [BBG+09], Haraka v2 [KLMR16], and others. These hash functions, collectively known as AES-like hashing, offer promising alternatives in cryptographic applications.

Received: 2023-09-01 Revised: 2023-11-23 Accepted: 2024-01-23 Published: 2024-03-01

At EUROCRYPT 2006, Kelsey and Kohno [KK06] introduced a new security property for hash functions called the chosen target forced-prefix (CTFP) preimage attack, also known as the Nostradamus attack. In this attack, the attacker knows the length of the prefix P and selects a public hash value h_T. The challenger then provides a prefix P, and the attacker is aiming to find a suffix S such that $h_T = H(P\|S)$. This attack resembles a hash-function-based commitment scheme, where the attacker uses h_T to commit a prediction of an event P in the future. Kelsey and Kohno [KK06] proposed the herding attack as the first Nostradamus attack, which is a generic attack applicable to any iterated hash function. For an n-bit output hash function, the herding attack can find a CTFP preimage with a short suffix in approximately $O(2^{2n/3})$ evaluations of the compression function.

Quantum Cryptanalysis. Besides the well-known quantum attacks on public-key cryptosystem RSA by Shor [Sho94], researchers have invented various quantum cryptanalysis techniques for symmetric ciphers over the past decade, such as the quantum polynomial-time attacks on 3-round Feistel [KM12], Even-Mansour construction [KM10], and various MACs and authenticated encryption schemes [KLLN16a], and more [LM17, BLNS21, BNS19a, DDW20, Hos22]. However, most attacks need to query the online quantum encryption oracles (known as Q2 model), which is believed to be impractical. Therefore, many researchers [BHN$^+$19, HS, KLLN16b, HS18, BNS19b, Sch23, CNS17] try to use only offline quantum computers to attack cryptosystems, where the plaintext-ciphertext pairs collected in classical ways (known as Q1 model). Different from block ciphers and other keyed primitives, the unkeyed hash functions can be implemented in offline quantum circuit and quantum superposition attacks [CNS17, HSX17, NS20, HS20, DSS$^+$20, HS21, SS22] can be freely applied to hash functions.

At ASIACRYPT 2022, Benedikt *et al.* [BFH22] examined the security of iterated hash functions against Nostradamus attacks by quantum attackers. They proposed generic quantum Nostradamus attacks, which involve a herding attack in the quantum setting, accelerating both offline and online phases when quantum random access memory (qRAM) is available. However, it is generally acknowledged that the difficulty of fabricating large qRAMs is enormous [GLM08, AGJ$^+$15], so quantum algorithms that use less or no qRAM (even with relatively high time complexity) are preferable [CNS17, NS20]. Therefore, Bao *et al.* [BGLP22] and Dong *et al.* [DLPZ23] proposed the low-qRAM version quantum herding attacks recently.

At ToSC 2023, Zhang *et al.* [ZSWH23] introduced the first dedicated Nostradamus attack utilizing the meet-in-the-middle (MITM) approach in both classical and quantum settings. Using an automatic tool, they successfully identified herding attack trails on reduced-round AES-MMO and Whirlpool that outperformed generic attacks proposed by Benedikt *et al.* [BFH22].

Multi-targets MITM and AES-like hashing. In 2010, Guo *et al.* [GLRW10] introduced a meet-in-the-middle attack that extends the framework to incorporate multi-target scenarios. This approach allows multiple available targets to provide additional degrees of freedom to one computation chunk without affecting the other. At FSE 2011, Sasaki [Sas11] was the first to employ MITM techniques to carry out preimage attacks on AES-like hash functions. At ToSC 2019, Bao *et al.* [BDG$^+$19] combined the multi-targets and MITM techniques to carry out pseudo-preimage attacks on AES hashing modes based on AES and Kiasu-BC.

Our contributions. In contrast to collision and preimage attacks, the resistance against the Nostradamus attack is seldom examined in iterated hash functions [KK06, BSU10, ABDK09]. Building upon recent advancements in quantum Nostradamus attacks on hash functions [BFH22, ZSWH23], we introduce a multi-target Meet-in-the-Middle Nostradamus

Table 1: Summary of our improved MITM Nostradamus attacks against the round-reduced AES-like hashing, compared with previous works

Target	Setting	#Rounds	Time	C-Mem	qRAM	Source
`AES-MMO`	Classical	6	$2^{82.7}$	$2^{82.2}$	-	[ZSWH23]
	Classical	6	2^{77}	2^{76}	-	Sect. 5.1
	Classical	7	2^{83}	2^{82}	-	Sect. 5.2
	Classical	any	$2^{88.1}$	$2^{87.8}$	-	[KK06, BSU10]
	Quantum	7	$2^{54.1}$	2^{14}	$2^{49.5}$ QRACM+2^{8} QRAQM	Sect. 5.3,[ZSWH23]
	Quantum	any	$2^{56.4}$	2^{17}	$2^{56.3}$ QRACM	[BFH22]
	Quantum	7	2^{58}	2^{30}	2^{8} QRAQM	Sect. 5.3
	Quantum	any	$2^{60.9}$	$2^{31.6}$	$O(n)$	[DLPZ23]
`Whirlpool`	Classical	4	2^{320}	2^{192}	-	[ZSWH23]
	Classical	6	2^{334}	2^{333}	-	Sect. 5.4
	Classical	any	$2^{344.7}$	$2^{344.2}$	-	[KK06, BSU10]
	Quantum	6	$2^{216.7}$	2^{64}	$2^{215.3}$ QRACM+2^{16} QRAQM	[ZSWH23]
	Quantum	6	2^{214}	2^{61}	$2^{207.4}$ QRACM+2^{24} QRAQM	Sect. 5.5
	Quantum	any	$2^{221.3}$	2^{71}	$2^{220.1}$ QRACM	[BFH22]
	Quantum	6	2^{230}	2^{117}	2^{24} QRAQM	Sect. 5.5
	Quantum	any	$2^{238.3}$	$2^{121.2}$	$O(n)$	[DLPZ23]

attack on AES-like hashing. This approach has the potential to provide incremental improvements to the results demonstrated by [ZSWH23]. By introducing the multi-targets technique, we have been able to attain higher rounds compared to previous results. Additionally, in line with Schrottenloher and Stevens' approach [SS22], we have transformed our attack into the quantum setting. Through this, we discovered that our quantum attack's time complexity can be enhanced while maintaining the same number of rounds. Interestingly, our multi-target technique differs slightly from the approach in the pseudo-preimage attack [BDG$^+$19]. Essentially, we enhance the freedom within the target, without limiting it to either forward or backward propagation.

At last, we propose enhanced meet-in-the-middle Nostradamus attack utilizing the adapted multi-targets technique. Building upon previous research on automatic tools for MITM attacks [BDG$^+$21, DHS$^+$21, BGST22, SS22, ZSWH23], we formulate a MILP (Mixed-Integer-Linear-Programming) model to discover an enhanced MITM Nostradamus attack targeting round-reduced AES-like hash functions. In classical settings, for both `AES-MMO` and `Whirlpool`, we improve the herding attacks by up to 2 rounds. In quantum settings, we significantly reduce the size qRAM, e.g., for 7-round `AES-MMO`, the qRAM is reduced from "2^8 QRAQM + $2^{49.5}$ QRACM" to "2^8 QRAQM", and for 6-round `Whirlpool` the qRAM is reduced from "2^{16} QRAQM + $2^{215.3}$ QRACM" to "2^{24} QRAQM". The results are summarized in Table 1.

Remark. Our results are presented in Table 1. It's worth noting that certain time and memory complexity values differ from those presented by [ZSWH23] for the same attacks. We will elaborate on the reasons for these differences in Section 3.1.

Outline. Section 2 provides a brief introduction to the target primitives, the generic Nostradamus attack, and the MITM preimage attack on AES-like hashing. The framework and main techniques of our MITM Nostradamus attack are summarized in Section 3, followed by the explanation of our MILP Model in Section 4. Results of classical and quantum Nostradamus MITM attacks are given in Section 5. Section 6 concludes the paper.

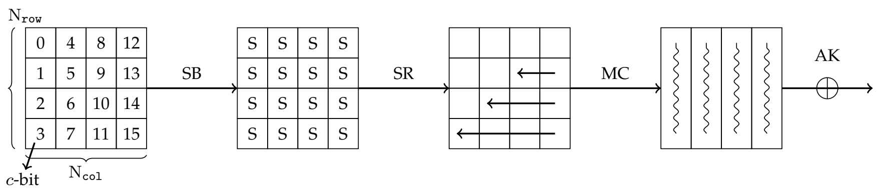


Figure 1: The `AES` round function

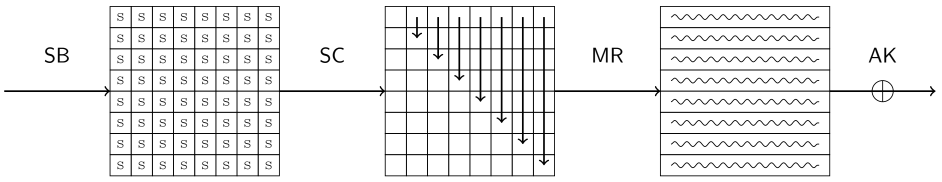


Figure 2: The `Whirlpool` round function

2 Preliminary

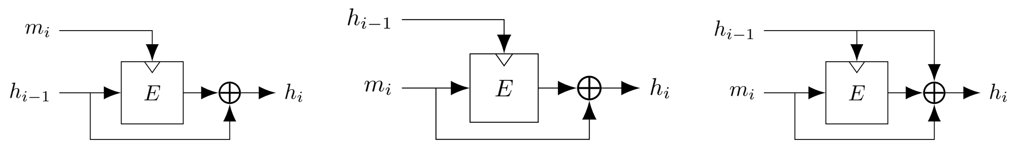
2.1 Hash functions based on `AES` and `Whirlpool`

`AES` is a block cipher family of 128-bit block and k-bit key for $k \in \{128, 192, 256\}$. The state has 16 bytes and can be represented as a 4×4 matrix. Given a $\mathrm{N_{row}} \times \mathrm{N_{col}}$ bytes of the state matrix, where $\mathrm{N_{row}} = \mathrm{N_{col}} = 4$, we order the byte cells as in Figure 1. Then the state is encrypted by an iterative process which is repeated for 10, 12, and 14 rounds, for `AES`-128, `AES`-192, and `AES`-256, respectively. An `AES` round function, as depicted in Figure 1, is an Substitution-Permutation Network (SPN), and composed of four consecutive operations: SubBytes (SB), ShiftRows (SR), MixColumn (MC), and AddRoundKey (AK). The master key k is (partially) XORed to the state before the initial round function application. This key is then utilized to generate r subkeys using the KeySchedule (KS) function. As our attack model does not require the key, we omit its specific details.

- `SubBytes` involves a nonlinear substitution that employs the same S-box for each byte of the internal state.
- `ShiftRow` is characterized by cyclically rotating the i-th row by i bytes to the left, where i takes values from 0 to 3.
- `MixColumn` entails multiplying each column by a Maximum Distance Separable (MDS) matrix over $\mathrm{GF}(2^8)$.
- `AddRoundKey` corresponds to performing an exclusive-OR operation with the round-dependent key.

Whirlpool [BR+00], as illustrated in Figure 2, is a hash function that draws a strong foundation from AES, albeit with an increased state size. Instead of ShiftRows (SR) and MixColumns (MC), it employs ShiftColumns (SC) and MixRows (MR). Whirlpool adopts an 8×8-byte (512-bit) state and is composed of 10 naturally extended AES rounds. The compression function output is generated using the MP mode, which is introduced below and depicted in Figure 3.

To construct hash functions based on block ciphers, a domain extension is required to apply the compression function iteratively. The widely employed domain extension in practice is the Merkle-Damgård approach [Dam90, Mer90]. This method involves padding the input message, denoted as M, such that the last block contains the original message length. The padded message is then divided into segments, $M_0\|M_1\|\dots\|M_{L-1}$, with each segment M_N being the length of a single block. An initial value H_0 is defined, and the iteration $H_N = CF(H_{N-1}, M_{N-1})$ is carried out for $N = 1, 2, \dots L$. Ultimately, H_L is produced as the hash value of M. The iteration function is built upon the DM mode, MMO mode, MP mode, etc. Let E_K denote a block cipher E with a key K. The construction process for each mode is illustrated in Figure 3 and expressed in following formulas.



(a) Davies-Meyer (DM) (b) Matyas-Meyer-Oseas (MMO) (c) Miyaguchi-Preneel (MP)

Figure 3: Hash modes

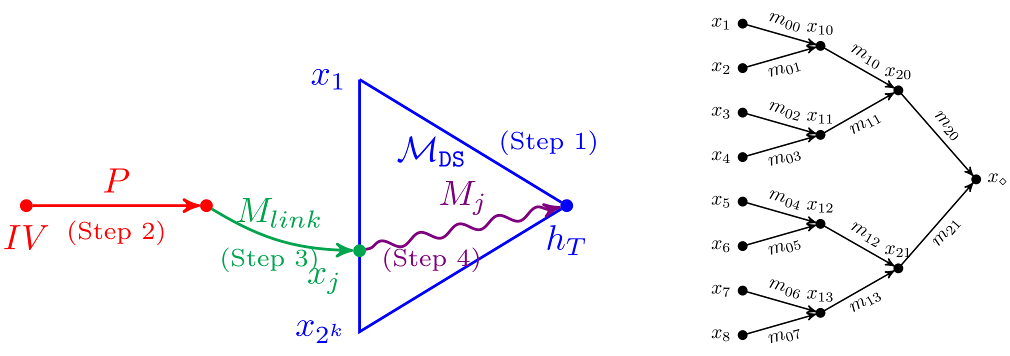
- **DM mode:** $CF(H_{N-1}, M_{N-1}) = E_{M_{N-1}}(H_{N-1}) \oplus H_{N-1}$.
- **MMO mode:** $CF(H_{N-1}, M_{N-1}) = E_{H_{N-1}}(M_{N-1}) \oplus M_{N-1}$.
- **MP mode:** $CF(H_{N-1}, M_{N-1}) = E_{H_{N-1}}(M_{N-1}) \oplus M_{N-1} \oplus H_{N-1}$.

2.2 Nostradamus Problem and Herding Attack

When Kelsey and Kohno introduced the Nostradamus problem [KK06], they also presented an elegant attack framework for general iterated hash functions, known as the "herding attack". The herding attack is composed of two phases. The offline phase involves calculating numerous chaining values and message blocks that ultimately result in the same hash value being committed. In the online phase, the goal is to identify the message block that can establish a connection between the chaining value from the chosen prefix and one of the chaining values acquired during the offline phase. The detailed algorithm could be illustrated as following:

- **Offline phase:** As an example given in Figure 4(b), build a diamond structure with 2^k leaves, k will be determined later to balance the time complexity of offline and online phases. Diamond is built in a way to find multi-collisions, each node x_i in the diamond represents a hash state or hash value, each edge m_j represents a message block so that two nodes x_{i_1}, x_{i_2} are connected by the edge m_j, and they satisfy the relation $CF(x_{i_1}, m_j) = x_{i_2}$. Kelsey and Kohno [KK06] present an algorithm which commences by sampling the leaf nodes and then progressively constructs the tree level by level. This is achieved by attempting various message blocks for each node in order to find collisions. Their algorithm requires approximately $2^{(n+k)/2}$ function evaluations. However, Blackburn *et al.* pointed out a flaw in the construction method and its complexity as presented in [KK06], subsequently providing a more meticulous analysis and construction method in [BSU10]. The approach presented in [BSU10] requires approximately $O(\sqrt{k} \cdot 2^{(n+k)/2})$ message blocks and $O(\sqrt{k} \cdot 2^{(n+k)/2})$ CF evaluations.
- **Online phase:** As shown in Figure 4(a), the online phase encompasses steps 2 to 4, commencing with the reception of the chosen prefix P. Initially, IV and P are used to determine the intermediate hash state x. In step 3, the key aspect of this online phase is to locate the connecting message M_{link} from x to one of the 2^k leaves x_j from the offline phase. This operation incurs a time cost of 2^{n-k}. In step 4, the objective is to search the stored diamond for the pathway from x_j to h_T, wherein

the message blocks M_j are retrieved. The suffix that fulfills the commitment h_T will be composed as $P\|M_{link}\|M_j$.



(a) Attack process [BGLP22] (b) A 2^3-diamond

Figure 4: Herding Attack

The quantum version of the herding attack is proposed in [BFH22], with acceleration occurring in both the offline and online phases. The quantum online phase is relatively straightforward, as it leverages the Grover algorithm to expedite the process from $O(2^{n-k})$ to $O(2^{(n-k)/2})$. On the other hand, the offline phase is quantized using the quantum collision-finding algorithm, reducing the time complexity from $O(2^{(n+k)/2})$ to $O(2^{(n+2k)/3})$, requiring quantum memory of $k^{-\frac{2}{3}} \cdot 2^{(n+2k)/3}$. This results in an overall balanced time complexity for the quantum herding attack of $O(2^{3n/7})$.

Recently, Bao *et al.* [BGLP22] designed a low-qRAM version of Benedikt *et al.*'s quantum herding attack. However, Dong *et al.* [DLPZ23] spotted Bao *et al.*'s algorithm is flawed and proposed a correct low-qRAM quantum herding attack based on CNS collision finding algorithm [CNS17]. Their time complexity to build the 2^k diamond structure is $k^{1/5} \cdot 2^{(2n+4k)/5}$ with a classical memory $k^{3/5} \cdot 2^{(n+2k)/5}$. The time complexity to find the M_{link} is $2^{\frac{n-r-k}{2}}(2^{r/2}+2^k)$ with $r \leq n$. Hence, the total time complexity is

$$2^{\frac{n-r-k}{2}}(2^{r/2}+2^k) + k^{1/5} \cdot 2^{(2n+4k)/5}. \tag{1}$$

The classical memory complexity is bounded by the construction of the diamond structure, i.e., $k^{3/5} \cdot 2^{(n+2k)/5}$. When $k = n/13$ and $r = 2n/13$, the optimal complexity is achieved which results in $O(2^{6n/13}) = O(2^{0.46n})$ time complexity and $O(2^{3n/13}) = O(2^{0.23n})$ classical memory.

2.3 The MITM Preimage Attack and Automation Framework

Brief Introduction. The MITM technique capitalizes on the loop structure determined by PGV modes and splits the computation of an encryption-based hash function into two chunks, forward and backward. The bytes involved in the encryption are categorized as:

- neutral bytes, values of which remain exclusive to the current chunk and exert no influence on the other. The forward neutral bytes are visualized as a blue cell 🟦, while the backward are visualized as red cells 🟥.
- constant bytes, values of which are pre-determined and known in both chunks, visualized as gray cells ⬛.

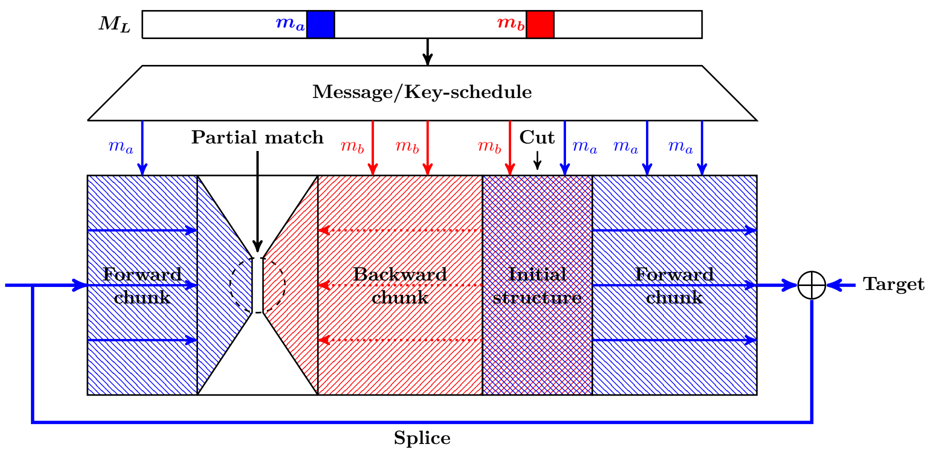


Figure 5: The latest MITM pseudo preimage attack framework [Sas11], "Target" depicted on the right side of the figure is to be regarded as "Multi-targets", as under discussion in this paper.

- arbitrary bytes, values of which are determined by both forward and backward neutral bytes, thus can be computed independently in neither chunk, visualized as white cells □.

The setting enables the computational independence of the two chunks. The two chunks meet in the middle at a shared intermediary state called the "matching point", where certain local constraints, namely partial-match constraints, are invoked to ensure the correctness of the enclosed computation. A MITM attack will filter the candidates by checking if the partial-match constraints are met before checking if it constitutes a valid preimage.

Complexity. As documented in various sources [BDG+19, BDG+21, BGST22, DHS+21], the standard MITM attack procedure encompasses the following steps:

1. Randomly assign values to the constant bytes.
2. Based on the values of the constant bytes, precompute the input for both chunks, denoted by N_+ for the forward chunk and N_- for the backward chunk. Assume $|N_+| = 2^{d_\mathcal{B}}$ and $|N_-| = 2^{d_\mathcal{R}}$.
3. For N_+, compute the forward chunk and store the value at the matching point in table T_+.
4. For N_-, compute the backward chunk and store the value at the matching point in table T_-.
5. Filter T_+ and T_- subject to partial-match constraints M. Assume $|M| = 2^{d_\mathcal{M}}$.
6. For an expected total of $2^{d_\mathcal{B}+d_\mathcal{R}-d_\mathcal{M}}$ candidates, check if it constitutes a match on full states.
7. Should a full match be identified, a preimage is determined. Otherwise, return to Step 1, alter the arbitrary values, and repeat Steps 2 to 6.

The computational complexity of this attack can be computed as demonstrated in [BDG+19]:

$$2^{n-(d_{\mathcal{B}}+d_{\mathcal{R}})} \cdot (2^{\max(d_{\mathcal{B}},d_{\mathcal{R}})} + 2^{d_{\mathcal{B}}+d_{\mathcal{R}}-d_{\mathcal{M}}}) \approx 2^{n-\min(d_{\mathcal{B}},d_{\mathcal{R}},d_{\mathcal{M}})}, \tag{2}$$

and the memory complexity would be $2^{\max(d_{\mathcal{B}},d_{\mathcal{R}})}$, but it can be reduced to $2^{\min(d_{\mathcal{B}},d_{\mathcal{R}})}$ by storing only the smaller table and computing the other chunk for matching.

Quantum Variant. The quantum Meet-in-the-Middle attack was initially introduced by Schrottenloher and Stevens [SS22]. They managed to achieve a quadratic speedup by utilizing quantum algorithm. We maintain the utilization of the same formula and model as presented in prior works, specifically [SS22, ZSWH23]. In this framework, when a meet-in-the-middle trail is defined by distinct blue, red, gray, and white cells, we can derive the degrees of freedom $d_{\mathcal{B}}, d_{\mathcal{R}}, d_{\mathcal{M}}$. As outlined in Theorem 3 in [ZSWH23], the time complexity of the quantum attack can be expressed as $2^{\frac{1}{2}(n-\min\{|d_{\mathcal{B}}-d_{\mathcal{R}}|,d_{\mathcal{B}},d_{\mathcal{R}},d_{\mathcal{M}}\})}$ which would be elaborated below.

The meet-in-the-middle attack procedure could be regarded as two-layer loop search, outer loop works on the constant bytes of the gray cells or step 1 of Procedure 2.3. After these bytes are fixed to some value, the inner loop would start from step 2 to 6 of Procedure 2.3. Use Theorem 1, the quantum time complexity of inner loop can be bounded by

$$2\{2|N_+| + (\frac{\pi}{4}\sqrt{|N_-|}+1)[\frac{\pi}{\sqrt{2}}\max(1,\sqrt{\frac{|N_+|}{|M|}})+6]\} \qquad \text{([SS22],Theorem 1)}$$

quantum evaluations of the attacked block cipher. Assume there is a single solution, then the iteration time of the outer loop will determined by the search space of constant bytes, i.e., $\frac{2^{n/2}}{\sqrt{|N_+|\cdot|N_-|}}$. Combining the outer and inner loop, the final quantum time complexity would be

$$\begin{aligned}2\frac{2^{n/2}}{\sqrt{|N_+|\cdot|N_-|}} \cdot \{2|N_+| + (\frac{\pi}{4}\sqrt{|N_-|}+1)[\frac{\pi}{\sqrt{2}}\max(1,\sqrt{\frac{|N_+|}{|M|}})+6]\}\\ = 2^{\frac{1}{2}(n-\min\{|d_{\mathcal{B}}-d_{\mathcal{R}}|,d_B,d_{\mathcal{R}},d_{\mathcal{M}}\})}.\end{aligned}$$

Theorem 1 ([BHMT02], Theorem 4)**.** *Let $\mathcal{A}$ be a quantum algorithm that uses no measurements, let $\chi : \mathbb{Z} \to \{0,1\}$ be a boolean function that tests if an output of $\mathcal{A}$ is "good". There exists a quantum algorithm that given the initial success probability $a > 0$ of $\mathcal{A}$, finds a good solution with certainty using a number of applications of $\mathcal{A}$ and $\mathcal{A}^{-1}$, which is in $\Theta(\frac{1}{\sqrt{a}})$ in the worst case.*

To achieve a realistic speedup, it's necessary to assume the availability of quantum memory for storing the values of the forward or backward chunks at the corresponding points to facilitate quantum lookup. In fact, quantum memory is also required for the offline constructed diamond structure.

qRAM. Quantum random access memory (qRAM) can be conceptualized as the quantum counterpart of classical random access memory (RAM). In the classical setting, RAM facilitates constant-time access (both read and write operations) to memory elements, regardless of storage size. qRAM also operates with constant-time access and is categorized into two types: quantum-accessible classical memory (QRACM), enabling access to classical data in quantum superpositions, and quantum-accessible quantum memory (QRAQM), wherein the data is stored within quantum memory. Consider a scenario where we intend to store a list of data, denoted as $D = (x_0, x_1, \cdots, x_{2^k-1})$, with each x_i representing an n-bit data. In this context, the qRAM for accessing the data D is established as a quantum gate. This qRAM is defined through a unitary operator $U_{qRAM}(D)$, which is expressed as follows:

$$U_{qRAM}(D) : |i\rangle\,|y\rangle \to |i\rangle\,|y \oplus x_i\rangle\,,$$

Here, i takes values from the set $\{0,1\}^k$, and y represents an n-bit value.

3 Meet-in-the-Middle Nostradamus Attack

3.1 Previous MITM Nostradamus Framework

In [ZSWH23], the MITM Nostradamus attack constructs the diamond structure using exactly the same method, requiring equivalent time and memory complexity for a given number of leaves 2^k. The shift occurs during the online phase, which transitions from exhaustive search to a different approach—either random sampling in the classical setting or Grover search in the quantum setting. Zhang *et al.* [ZSWH23] employ the MITM framework to search for linking message blocks, progressing from the prefix to one of the leaves. In brief, by using the prefix as the key and the leaves as the multiple images in AES-like MMO/MP hashing function, the MITM online phase is then employed to locate a single preimage. This approach yields improved time complexity, particularly when the degrees of freedom in terms of forward, backward, and matching chunks are thoughtfully chosen.

Specifically, the time complexity of the online phase is $2^{n-\min\{d_{\mathcal{B}},d_{\mathcal{R}},d_{\mathcal{M}}\}}$ (or respectively, $2^{\frac{1}{2}(n-\min\{|d_{\mathcal{B}}-d_{\mathcal{R}}|,d_{\mathcal{B}},d_{\mathcal{R}},d_{\mathcal{M}}\})}$), where $d_{\mathcal{B}}, d_{\mathcal{R}}, d_{\mathcal{M}}$ correspond to the degrees of freedom for blue or forward chunks, red or backward chunks, and matching chunks in the classical (quantum) setting. The general time complexity is 2^{n-k} ($2^{\frac{1}{2}(n-k)}$) in the classical (quantum) setting. As a result, the values of $d_{\mathcal{B}}, d_{\mathcal{R}}, d_{\mathcal{M}}, k$ need to be constrained in such a way that the dedicated time complexity outperforms the general time complexity. When integrating the MITM framework into the MILP model, these restrictions are replaced with their respective constraints.

Time Complexity. Here, we will discuss time complexity calculation proposed by Zhang *et al.* [ZSWH23]. Suppose Benedikt *et al.*'s quantum generic time complexity $O(\sqrt[3]{k} \cdot 2^{(n+2k)/3} + 2^{(n-k)/2})$ can be approximated to $b_q(n,k) = \sqrt[3]{k} \cdot 2^{(n+2k)/3} + 2^{(n-k)/2}$, the omission of big O notation includes the exclusion of constant factors such as $\pi/4$ from Grover's algorithm. Zhang *et al.* reference the results of time complexity established by Benedikt *et al.* and claim that, the generic Nostradamus attack's time complexity on any round of AES is $2^{88.8}$ in the classical setting and $2^{57.2}$ in the quantum setting. However, we have found that these results were computed using further simplified formulas, namely, $O(\sqrt[3]{n} \cdot 2^{3n/7})$ in the quantum setting, with $k = n/7$ substituted into the formula above. However, when we evaluate $b_q(n, n/7)$, we find that it equals $(\sqrt[3]{\frac{n}{7}} + 1) \cdot 2^{3n/7}$, and for the specific case of $n = 128$, $b_q(128, 128/7)$ results in $2^{56.7}$. It's worth noting that for the function $b_q(n,k)$, using a smaller k can lead to a reduction in its value. This adjustment helps in achieving a more balanced outcome between the two terms. As an illustration, when k is set to 17, $b_q(128, 17) = \sqrt[3]{17} \times 2^{54} + 2^{55.5} = 2^{56.4}$. When they introduced a 7-round quantum Nostradamus attack on AES-MMO, the total time complexity was 2^{56}, where 2^{56} for the online phase and 2^{50} for the offline phase. The time complexity is remarkably close to our calculated balanced generic complexity of $2^{56.4}$, as opposed to their claim of $2^{57.2}$. This raises concerns about the validity of their 7-round attack.

Similarly, we can calculate the classical time complexity of a generic Nostradamus attack, which is $b_c(n,k) = \sqrt[2]{k} \cdot 2^{(n+k)/2} + 2^{n-k}$. This achieves its smallest value of $2^{88.1}$ when $k = 42$.

As for `Whirlpool`, the specific generic time complexity in the classical setting would be calculated as $b_c(512, 169) = 2^{344.7}$ when $k = 169$, and the quantum complexity would be $b_q(512, 71) = 2^{221.3}$ when $k = 71$.

In regards to the unit of time complexity, it is imperative to acknowledge that the time complexity of the Nostradamus attack is gauged by the number of function evaluations, specifically the time required to evaluate a particular instance of AES-like hashing in both classical and quantum settings.

Memory Complexity. In both [BGLP22] and [ZSWH23], the memory complexity are cited incorrectly. In the context of classical attacks, the algorithms introduced by [KK06] and [BSU10] construct a diamond structure of size 2^k. This approach comes with a memory complexity of $O(2^{(n+k)/2})$, as each node necessitates $O(2^{(n-k)/2})$ messages that need to be stored and sorted. For the quantum attack algorithm presented by [BFH22], a diamond structure is constructed for half of the leaves in similar manner. Each node in this structure invokes $O(2^{(n-k)/3})$ hash evaluations to balance time complexity, resulting in a QRACM complexity of $O(2^{(n+2k)/3})$. This complexity aligns with the time complexity, irrespective of constant factors. The accurate memory complexity is presented in Table 1.

Regarding the unit of memory complexity, it is noteworthy that the predominant factor of the memory complexity is the memory used in the diamond structure part, responsible for storing triplets $(m_j, r, CF(x_{i_1}, m_j))$, as explicated in Section 2.2. Consequently, the unit for memory complexity becomes the triplet length, or O(7n/3) bits.

3.2 Modified Multi-target MitM Nostradamus Framework

While reviewing the MITM trails presented in [ZSWH23], we have identified room for improvement in their analysis of freedom. To capitalize on the potential of multiple targets offered by the diamond structure, the authors of the study limit the cells in the target state to red, blue, or gray. This categorization allows red and blue cells to serve as multi-target parameters. In this particular scenario, the degrees of freedom of $d_{\mathcal{B}}, d_{\mathcal{R}}$ are calculated from $d_{\mathcal{B}}^{\texttt{ENC}} + d_{\mathcal{B}}^{\texttt{TAG}}$, and $d_{\mathcal{R}}^{\texttt{ENC}} + d_{\mathcal{R}}^{\texttt{TAG}}$. Here, $d_{\mathcal{B}}^{\texttt{ENC}}$ ($d_{\mathcal{R}}^{\texttt{ENC}}$) represents the count of blue (red) cells in the initial state of the encryption computation path, *i.e.*, $\#\mathrm{SR}^4$ in Figure 6, while $d_{\mathcal{B}}^{\texttt{TAG}}$ ($d_{\mathcal{R}}^{\texttt{TAG}}$) signifies the number of blue (red) cells in the target state, *i.e.*, bottom state "original" or "new" in Figure 6.

Let's consider an extreme scenario where blue or red cells in the target state could be freely substituted with an equivalent number of white cells. In other words, as illustrated in Figure 6, modifying the target state in this manner doesn't impact the colors of any other state within the differential trail. In this case, the time complexity transitions to $2^{n-d_{\mathcal{W}}-\min\{d_{\mathcal{B}}^{\texttt{ENC}}, d_{\mathcal{R}}^{\texttt{ENC}}+d_{\mathcal{R}}^{\texttt{TAG}}, d_{\mathcal{M}}\}}$ in the classical setting and equals to $2^{n-d_{\mathcal{B}}^{\texttt{TAG}}-\min\{d_{\mathcal{B}}^{\texttt{ENC}}, d_{\mathcal{R}}^{\texttt{ENC}}+d_{\mathcal{R}}^{\texttt{TAG}}, d_{\mathcal{M}}\}}$, where $d_{\mathcal{W}}$ represents the degree of freedom of white cells in the target state. Notably, the complexity is less than or equal to the original complexity $2^{n-\min\{d_{\mathcal{B}}^{\texttt{ENC}}+d_{\mathcal{B}}^{\texttt{TAG}}, d_{\mathcal{R}}^{\texttt{ENC}}+d_{\mathcal{R}}^{\texttt{TAG}}, d_{\mathcal{M}}\}}$. Therefore, introducing white cells could potentially lead to the discovery of better differential trails for MITM attacks.

Our novel model takes into account a scenario where blue, red, gray, and white cells could appear simultaneously in the target state, representing a generalized version of [ZSWH23]. If we denote k to be the number of targets employed for the MITM attack, which should be less than or equal to $d_{\mathcal{W}}$, the time complexity would be $2^{n-k-\min\{d_{\mathcal{B}}^{\texttt{ENC}}+d_{\mathcal{B}}^{\texttt{TAG}}, d_{\mathcal{R}}^{\texttt{ENC}}+d_{\mathcal{R}}^{\texttt{TAG}}, d_{\mathcal{M}}\}}$ in the classical setting and $2^{\frac{1}{2}(n-k-\min\{|d_{\mathcal{B}}^{\texttt{ENC}}+d_{\mathcal{B}}^{\texttt{TAG}}-d_{\mathcal{R}}^{\texttt{ENC}}-d_{\mathcal{R}}^{\texttt{TAG}}|, d_{\mathcal{B}}^{\texttt{ENC}}+d_{\mathcal{B}}^{\texttt{TAG}}, d_{\mathcal{R}}^{\texttt{ENC}}+d_{\mathcal{R}}^{\texttt{TAG}}, d_{\mathcal{M}}\})}$ in the quantum setting.

The size of the diamond structure would now be $2^{k+d_{\mathcal{B}}^{\texttt{TAG}}+d_{\mathcal{R}}^{\texttt{TAG}}}$, resulting in a time

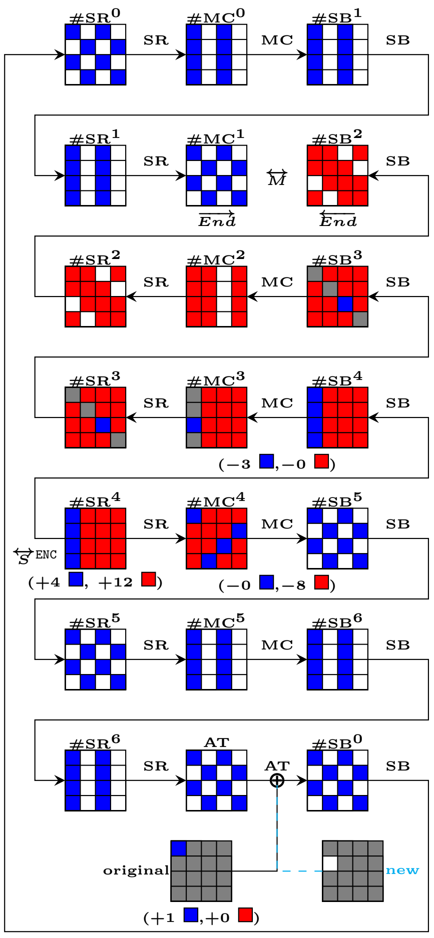


Figure 6: Substitute the target state in Figure 8 [ZSWH23] with an updated configuration

complexity of building the diamond structure as $2^{\frac{n+k+d_{\mathcal{B}}^{\mathtt{TAG}}+d_{\mathcal{R}}^{\mathtt{TAG}}}{2}}$, while the quantum time would be $2^{\frac{n+2(k+d_{\mathcal{B}}^{\mathtt{TAG}}+d_{\mathcal{R}}^{\mathtt{TAG}})}{3}}$. The parameter k is chosen within the range of 0 to $d_{\mathcal{W}}$ to strike a balance in the time complexity of the two phases.

Another important difference would be the elimination of the condition of

$$|d_{\mathcal{B}} - d_{\mathcal{R}}| \geq \min\{d_{\mathcal{B}}, d_{\mathcal{R}}, d_{\mathcal{M}}\},$$

since our viewpoint is that the necessity for the quantum attack to achieve quadratic acceleration isn't absolute. The sole criterion for assessing the validity of the quantum attack is that its time complexity is lower than the corresponding generic bound.

Remark 1. In all the MITM differential trails we found for herding attacks on dedicated ciphers like `AES-MMO` and `Whirlpool`, there are, in fact, no blue and red cells in the target state, *i.e.*, $d_{\mathcal{B}}^{\mathtt{TAG}} = d_{\mathcal{R}}^{\mathtt{TAG}} = 0$. Instead, the target state consists only of gray and white cells. The reasons for the optimization results produced by automated tools require further analysis.

In our analysis of the approach to identifying potential Nostradamus attacks on `AES`-like hashing, it was observed that the parameter $d^{\mathtt{TAG}}$ does not plays a significant role since the presented paths showed $d_{\mathcal{B}}^{\mathtt{TAG}} = d_{\mathcal{R}}^{\mathtt{TAG}} = 0$. This is in contrast to previous models discussed in [ZSWH23], where $d^{\mathtt{TAG}}$ was considered indispensable due to multi-target requirement.

Focusing solely on the online phase for searching linking messages in a classical setting, the dedicated MitM approach exhibits a time complexity of $2^{n-\min(d_{\mathcal{B}}^{\texttt{ENC}}+d_{\mathcal{B}}^{\texttt{TAG}},d_{\mathcal{R}}^{\texttt{ENC}}+d_{\mathcal{R}}^{\texttt{TAG}},d_{\mathcal{M}})}$, which proves more efficient than the generic time complexity of $2^{n-(d_{\mathcal{B}}^{\texttt{TAG}}+d_{\mathcal{R}}^{\texttt{TAG}})}$ under the condition that $\min(d_{\mathcal{B}}^{\texttt{ENC}}+d_{\mathcal{B}}^{\texttt{TAG}},d_{\mathcal{R}}^{\texttt{ENC}}+d_{\mathcal{R}}^{\texttt{TAG}},d_{\mathcal{M}}) > d_{\mathcal{B}}^{\texttt{TAG}}+d_{\mathcal{R}}^{\texttt{TAG}}$. This condition implies that $d_{\mathcal{B}}^{\texttt{ENC}} > d_{\mathcal{R}}^{\texttt{TAG}}$, $d_{\mathcal{R}}^{\texttt{ENC}} > d_{\mathcal{B}}^{\texttt{TAG}}$, and $d_{\mathcal{M}} > d_{\mathcal{B}}^{\texttt{TAG}}+d_{\mathcal{R}}^{\texttt{TAG}}$. However, to identify a valid Nostradamus attack when the generic time complexity T is known, two additional constraints must be considered: $2^{n-\min(d_{\mathcal{B}}^{\texttt{ENC}}+d_{\mathcal{B}}^{\texttt{TAG}},d_{\mathcal{R}}^{\texttt{ENC}}+d_{\mathcal{R}}^{\texttt{TAG}},d_{\mathcal{M}})} < T$ and $\sqrt[2]{d_{\mathcal{B}}^{\texttt{TAG}}+d_{\mathcal{R}}^{\texttt{TAG}}} \cdot 2^{(n+d_{\mathcal{B}}^{\texttt{TAG}}+d_{\mathcal{R}}^{\texttt{TAG}})/2} < T$.

Remark 2. A valid MITM path for preimage attack may not necessarily result in a valid Nostradamus attack, but the presence of a sufficient number of white bytes in the target state within the path make it a potential candidate for a MITM preimage attack.

4 MILP Model

This section describes how the search for MITM Nostradamus attacks is automated with Mixed-Integer Linear Programming (MILP). The source code of our model is available at `https://github.com/shahuiz/MITM-Nostradamus`.

4.1 Automation Framework

An overview of the automatic search framework is provided before the modeling details are illustrated. As the degree of freedom from the key schedule is not to be efficiently exploited in the attack [ZSWH23], the framework is built on top of a conventional automated MITM attack in a single-key setting with a twist on the optimization objective.

The special states in a MITM attack are identified using the following notations:

- $\overleftrightarrow{S}^{\texttt{ENC}}$: the starting encryption state for forward and backward chunks.
- $\overleftrightarrow{S}^{\texttt{TAG}}$: the target state for forward and backward chunks.
- $\overrightarrow{End}$: the terminating state of the forward chunk.
- $\overleftarrow{End}$: the terminating state of the backward chunk.
- $\overleftrightarrow{M}$: the matching round operator between the two terminating states.

The positions of the aforementioned special states determine different patterns of segmenting the closed loop. By default, the patterns are specified in a round-level accuracy. The attack configuration parameter `config` is defined as the ordered tuple with the following attributes:

- `TOTAL`: the total attacked rounds.
- `START`: the round index of $\overleftrightarrow{S}^{\texttt{ENC}}$.
- `MATCH`: the round index of $\overleftrightarrow{M}$, $\overrightarrow{End}$, and $\overleftarrow{End}$.

In our attack, initial degrees of freedom (DOFs) could be originated from the state as well as the target. The bytes in $\overleftrightarrow{S}^{\texttt{ENC}}$ and $\overleftrightarrow{S}^{\texttt{TAG}}$ are thus partitioned into pairwise disjoint subsets with different coloring (i.e. blue, red, gray, and white). The subsets are named after the first letter of the coloring with source superscripted: $\mathcal{B}^{\texttt{ENC}}$, $\mathcal{R}^{\texttt{ENC}}$, $\mathcal{G}^{\texttt{ENC}}$, $\mathcal{W}^{\texttt{ENC}}$ and $\mathcal{B}^{\texttt{TAG}}$, $\mathcal{R}^{\texttt{TAG}}$, $\mathcal{G}^{\texttt{TAG}}$, $\mathcal{W}^{\texttt{TAG}}$. $\overrightarrow{\iota}$ and $\overleftarrow{\iota}$ are used to denote the initial DOFs of forward and backward computations. Additional constraints may be added during the attribute propagation of each chunk to negate mutual impact and retain computational independence, which will be reflected in the consumption of the initial DOFs. We use $\overrightarrow{\sigma}$ and $\overleftarrow{\sigma}$ to track the consumed

DOFs. The remaining DOFs at the end of each computation are denoted as $d_{\mathcal{B}}$ and $d_{\mathcal{R}}$. Intuitively, we have:

$$\overrightarrow{\iota} = |\mathcal{B}^{\texttt{ENC}}| + |\mathcal{B}^{\texttt{TAG}}| \quad , \quad \overleftarrow{\iota} = |\mathcal{R}^{\texttt{ENC}}| + |\mathcal{R}^{\texttt{TAG}}| \tag{3}$$

$$d_{\mathcal{B}} = \overrightarrow{\iota} - \overrightarrow{\sigma} \quad , \quad d_{\mathcal{R}} = \overleftarrow{\iota} - \overleftarrow{\sigma} \tag{4}$$

The degree of matching $d_{\mathcal{M}}$ is determined by the choice of $\overleftrightarrow{M}$ and the byte distribution of $\overrightarrow{End}$ and $\overleftarrow{End}$. If MATCH locates at the last round, $\overleftrightarrow{M}$ is the XOR operator of the feedforward step. Otherwise, $\overleftrightarrow{M}$ is selected to be a MixColumns operator. We define the number of bits in a byte as a constant denoted by NBYTE, which equals 8. In the classic setting, the MITM attack would outperform brute force search (in the logarithm of 2) by the factor of τ^C with the formula:

$$\tau^C = \texttt{NBYTE} \cdot \min(d_{\mathcal{B}}, d_{\mathcal{R}}, d_{\mathcal{M}}) \tag{5}$$

In the quantum setting, the speed up τ^Q in comparison with a plain Grover's search over the preimage space is also determined by the imbalance between blue and red remaining degrees:

$$\tau^Q = \texttt{NBYTE} \cdot \frac{\min(|d_{\mathcal{B}} - d_{\mathcal{R}}|, d_{\mathcal{B}}, d_{\mathcal{R}}, d_{\mathcal{M}})}{2} \tag{6}$$

The Guess-and-Determine (GnD) technique proposed in [BGST22] is incorporated into the classic attack framework. In cases where some white cells ruined information at the MixColumns operator, GnD provides an alternative to guess the byte to fulfill the requirements for propagation and compensate with costs. In the search for extended-round attacks on ciphers with large intermediate states, GnD proved its superiority when the significance of successful propagation outweighed the cost. We use $g_{\mathcal{B}}$ to count the guessed blue bytes, $g_{\mathcal{R}}$ the guessed red bytes, and $g_{\mathcal{BR}}$ the guessed linear composition of both blue and red bytes. The advantage that the GnD-integrated MITM attack brings in comparison to brute force (in the logarithm of 2) is denoted by $\tau^{\texttt{GnD}}$ with the formula:

$$\tau^{\texttt{GnD}} = \texttt{NBYTE} \cdot \min(d_{\mathcal{B}} - g_{\mathcal{R}}, d_{\mathcal{R}} - g_{\mathcal{B}}, d_{\mathcal{M}} - g_{\mathcal{B}} - g_{\mathcal{R}} - g_{\mathcal{BR}}) \tag{7}$$

A conventional single-key single-target MITM framework is concluded above. In our proposed multi-target setting, the online MITM phase could be accelerated by searching for a partial match instead of a full-state match. We use k_w to denote the number of bits that could be arbitrary during the partial match:

$$k_w \leq \texttt{NBYTE} \cdot |\mathcal{W}^{\texttt{TAG}}| \tag{8}$$

The offline phase of the Nostradamus MITM attack builds a diamond structure for the herding attack, the height of which is denoted by k_d (in the logarithm of 2). The value of k_d is determined by the number of partial targets obtained during the online MITM phase. In the multi-target setting, both the arbitrary bits and the active bytes in the target contributes to k_d. Thus, we have:

$$k_d = k_w + \texttt{NBYTE} \cdot (|\mathcal{B}^{\texttt{TAG}}| + |\mathcal{R}^{\texttt{TAG}}|) \tag{9}$$

The overall performance of the attack should be determined by the balance of offline and online phases. The optimization objective will be discussed in both classic and quantum settings. We denote the block length of the cipher as N.

Classic Objective C is formulated as follows:

$$C = \max(N - k_w - \tau^{\texttt{GnD}}, \quad (N + k_d)/2) \tag{10}$$

In addition, to yield a meaningful attack that outperforms the corrected generic bound given in section 3.1, we add:

$$k_w + \tau^{\texttt{GnD}} > k_d \tag{11}$$

Quantum Objective Q is formulated as follows:

$$Q = \max((N - k_w)/2 - \tau^Q, \quad (N + 2 \cdot k_d)/3) \tag{12}$$

Similarly, we also add:

$$k_w/2 + \tau^Q > k_d/2 \tag{13}$$

4.2 MILP Modeling

In this subsection, we dive into the details of how to model the attack with the MILP language.

4.2.1 Encoding Scheme

We model the intermediate states with three boolean variables: b, r, and w.

- A blue byte is encoded as $(b, r, w) = (1, 0, 0)$
- A red byte is encoded as $(b, r, w) = (0, 1, 0)$
- A white byte is encoded as $(b, r, w) = (0, 0, 1)$
- A gray byte is encoded as $(b, r, w) = (0, 0, 0)$

We also use the notations $b_\alpha, r_\alpha, w_\alpha$ to represent the value of b, r, w of byte α. The encoding scheme enables us to efficiently obtain $|\mathcal{B}^{\texttt{ENC}}|$, $|\mathcal{R}^{\texttt{ENC}}|$, $|\mathcal{B}^{\texttt{TAG}}|$, $|\mathcal{R}^{\texttt{TAG}}|$, $|\mathcal{W}^{\texttt{TAG}}|$ by simply summing up the corresponding values of coordinates.

4.2.2 Propagation Rules

Taking the example of `AES` for illustration, it's worth noting that `Whirlpool` follows similar principles.

`SubBytes` The operator is an identity transformation. Such construction could be easily obtained by the convex hull method, which is a common and well-referenced technique in MILP-based automation.

`XOR` Due to the single-key setting, the `XOR` operators are only deployed at the last round, where the output is mixed with the target. An `XOR` operator input 2 bytes and output 1 byte. The rule is described as follows:

- When the input involves a white byte, the output is white with no DOF consumption.
- When the input contains only gray bytes, the output is gray with no DOF consumption.
- When the input contains only blue bytes, the output is either blue with no DOF consumption or gray with 1 DOF consumption from the forward chunk.($\overrightarrow{\sigma} = \overrightarrow{\sigma} + 1$)
- When the input contains only red bytes, the output is either red with no DOF consumption or gray with 1 DOF consumption from the backward chunk.($\overleftarrow{\sigma} = \overleftarrow{\sigma} + 1$)
- When the input is a mix of red and blue bytes, the output is white:

The rule requires two extra binary variables at each output position to track the cost of DOF and could be converted to constraints by the convex-hull method.

`MixColumns` A `MixColumns` operator inputs and outputs a column. We denote the cardinality of a column as `NROW`, which equals 4. We could compose the basic rule for the `MixColumns` operator by introducing two extra integer-valued variables to track the cost of DOF, which is detailed as follows:

- When the input contains a white byte, the output contains only white bytes with no consumption of DOF.
- When the input contains only gray, the output contains only gray bytes with no consumption of DOF.
- When the input is a mix of i blue bytes, j red bytes, k gray bytes ($k < \texttt{NROW}$), l white bytes, the output will be a mix of a blue bytes, b red bytes, c gray bytes and d white bytes, with $a+c$ consumed DOF from the backward chunk and $b+c$ consumed DOF from the forward chunk. The relation is formulated as:

$$i+j+k = a+b+c+d = \texttt{NROW}, \quad a+c \leq \texttt{NROW}\cdot(j+l), \quad b+c \leq \texttt{NROW}\cdot(i+l) \quad (14)$$

`GnD-MC` The construction of `MixColumns` operator with GnD integrated is organized as follows. Conceptually, GnD is implemented as an independent operator, leading to the propagation: $A \to \text{GnD} \to B \to \text{MC} \to C$, where A, B, C represent intermediate states. Through our optimized construction, we can bypass the intermediary state B and directly transform A to C using a `GnD-MC` operator. To achieve this objective, we introduced 4 additional GnD switches for an input byte α, namely g^w_α, g^b_α, g^r_α, g^{br}_α. The switches are binary that satisfy:

$$w_\alpha = g^w_\alpha + g^b_\alpha + g^r_\alpha + g^{br}_\alpha \quad (15)$$

With the incorporation of only 4 additional GnD switches, we achieve the realization of GnD-MC on top of basic `MixColumns`. Notably, this construction allows for convenient toggling of GnD on and off as required.

In this setting, if α is non-white, then all GnD switches are 0. Otherwise, exactly one GnD switch equals 1 while the rest are 0: $g^w_\alpha = 1$ means GnD is not activated and byte α remains unknown, $g^b_\alpha = 1$ means byte α is guessed as blue for forward propagation, $g^r_\alpha = 1$ means byte α is guessed as red for backward propagation, and $g^{br}_\alpha = 1$ means byte α is guessed white for both forward and backward propagation. Hence, the column becomes a mix of $\sum_\alpha g^w_\alpha$ white bytes, $\sum_\alpha g^b_\alpha + b_\alpha$ blue bytes, $\sum_\alpha g^r_\alpha + r_\alpha$ red bytes, $\sum_\alpha g^{br}_\alpha$ white bytes, and $\texttt{NROW} - \sum_\alpha(b_\alpha + r_\alpha + w_\alpha)$ gray bytes after GnD. With the new byte distribution, we could call the basic model to properly address the propagation.

$\overleftrightarrow{M}$ There are two types of matching equipped in our attack.

The `XOR`-match is used in the last round around the feedforward. It checks $\overrightarrow{End}$, $\overleftarrow{End}$, and the target `TAG` byte by byte, and $m_\alpha = 1$ if $w_\alpha = 0$ for $\overrightarrow{End}$, $\overleftarrow{End}$, and `TAG`. The final $d_\mathcal{M}$ is the summation over all m_αs:

$$d^{\texttt{XOR}}_\mathcal{M} = \sum_\alpha m_\alpha \quad (16)$$

We adopt the `MC`-match at intermediate rounds around a `MixColumns` operator. It checks $\overrightarrow{End}$ and $\overleftarrow{End}$ and counts the non-white bytes column by column. For a column β, if there exists $t_\beta > \texttt{NROW}$ non-white bytes, then we have a degree of matching $t_\beta - \texttt{NROW}$ at column β. The final $d_\mathcal{M}$ could be formulated as follows:

$$d^{\texttt{MC}}_\mathcal{M} = \sum_\beta \max(0, t_\beta - \texttt{NROW}) \quad (17)$$

5 Main Results on AES and Whirlpool

In this section, we use the notations as follows. The state before the SubByte operation of the r^{th} round is denoted as $\#\text{SB}^r$. Similarly, the state before ShiftRow or ShiftColumn operation of r^{th} round is denoted as $\#\text{SR}^r$ or $\#\text{SC}^r$, the state before MixColumn or MixRow operation of r^{th} round is denoted as $\#\text{MC}^r$ or $\#\text{MR}^r$, the state before adding target is denoted as AT. And we denote the byte at the row i and column j of a state X as $X_{\texttt{NROW}\cdot j+i}$, where `NROW` is 4 in the context of `AES` and 8 in the context of `Whirlpool`.

5.1 Improved 6-round attack on AES-MMO

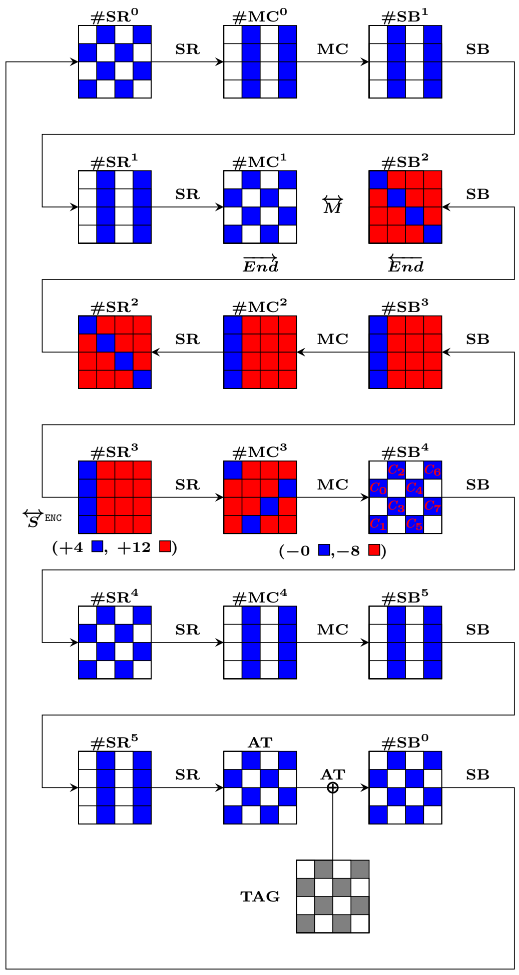


Figure 7: The 6-round attack on `AES-MMO` in classical setting

The attack starts with the precomputation of blue and red initial values. Recall that during propagation, constraints are imposed on certain cells to preserve propagation trails, represented by the consumption of DOF. An MITM attack fixes the value of such beforehand and precomputes the initial values satisfying those constraints.

We have equivalently chosen $\#\text{MC}^3$ as the initial state for both forward and backward chunks. And we have the input of the forward chunk $\mathcal{B}^{\texttt{ENC}}$ as $\{\#\text{MC}^3_0, \#\text{MC}^3_7, \#\text{MC}^3_{10}, \#\text{MC}^3_{13}\}$ and the input of the backward chunk $\mathcal{R}^{\texttt{ENC}} = \#\text{MC}^3 \backslash \mathcal{B}^{\texttt{ENC}}$.

Precomputation of blue initial values. Given that there are no restrictions applied to the forward chunk, we have the freedom to select the values of $\#\mathrm{MC}^3_0$, $\#\mathrm{MC}^3_7$, $\#\mathrm{MC}^3_{10}$, $\#\mathrm{MC}^3_{13}$ and then propagate them to the corresponding states $\#\mathrm{MC}^1$ and $\#\mathrm{SB}^2$. The input space of blue bytes has a size of $2^{4\times 8} = 2^{32}$.

Precomputation of red initial values. A total of 8 constraints are placed on the input space of the backward chunk at the `MixColumns` operator in round 3. To illustrate, let's consider column 0: $\#\mathrm{MC}^3_1$, $\#\mathrm{MC}^3_2$, and $\#\mathrm{MC}^3_3$ should have a constant impact on $\#\mathrm{SB}^4_1$ and $\#\mathrm{SB}^4_3$ to maintain the independence of the forward computation. As a result, the constraints for column 0 can be outlined as follows, with c_0 and c_1 representing predefined constants:

$$\begin{aligned} 2\cdot\#\mathrm{MC}^3_1 + 3\cdot\#\mathrm{MC}^3_2 + 1\cdot\#\mathrm{MC}^3_3 &= c_0 \\ 1\cdot\#\mathrm{MC}^3_1 + 1\cdot\#\mathrm{MC}^3_2 + 2\cdot\#\mathrm{MC}^3_3 &= c_1 \end{aligned} \tag{18}$$

Likewise, constraints can be imposed on the remaining red bytes within $\#\mathrm{MC}^3$ based on predefined constants c_2, c_3, c_4, c_5, c_6, and c_7. Ultimately, this leads us to an input space of size $2^{4\times 8} = 2^{32}$ for the backward chunk.

Match between the matching states. The forward and backward computations conclude at $\#\mathrm{MC}^1$ and $\#\mathrm{SB}^2$, and the matching process will be conducted column by column. For example, in column 0, we have obtained $\#\mathrm{MC}^1_1$, $\#\mathrm{MC}^1_3$, and $\#\mathrm{SB}^2_0$ during the forward computation, and $\#\mathrm{SB}^2_1$, $\#\mathrm{SB}^2_2$, and $\#\mathrm{SB}^2_3$ during the backward computation. If a partial match is present for column 0, it should adhere to the following:

$$\begin{cases} \#\mathrm{MC}^1_1 = 9\cdot\#\mathrm{SB}^2_0 + 14\cdot\#\mathrm{SB}^2_1 + 11\cdot\#\mathrm{SB}^2_2 + 13\cdot\#\mathrm{SB}^2_3 \\ \#\mathrm{MC}^1_3 = 11\cdot\#\mathrm{SB}^2_0 + 13\cdot\#\mathrm{SB}^2_1 + 9\cdot\#\mathrm{SB}^2_2 + 14\cdot\#\mathrm{SB}^2_3 \end{cases} \tag{19}$$

which leads to

$$\begin{cases} \#\mathrm{MC}^1_1 - 9\cdot\#\mathrm{SB}^2_0 = 14\cdot\#\mathrm{SB}^2_1 + 11\cdot\#\mathrm{SB}^2_2 + 13\cdot\#\mathrm{SB}^2_3 \\ \#\mathrm{MC}^1_3 - 11\cdot\#\mathrm{SB}^2_0 = 13\cdot\#\mathrm{SB}^2_1 + 9\cdot\#\mathrm{SB}^2_2 + 14\cdot\#\mathrm{SB}^2_3 \end{cases} \tag{20}$$

Likewise, for columns 1, 2, and 3, an additional 6 equations would be necessary to maintain the partial match. This would result in a filter of probability $2^{8\times 8} = 2^{64}$.

Attack Procedure. The meet-in-the-middle Nostradamus attack procedure is executed in the following manner:

1. Set the values of the 64-bit gray cells and 44-bit white cells in the target to zeros. Construct the diamond structure with 2^{20} leaves which follows the configuration of the target, necessitating a time complexity of $\sqrt{20} \times 2^{(128+20)/2} = 2^{76.2}$ and a memory complexity of approximately $0.83 \times 2^{76.2}$ (the coefficient 0.83 is derived from [BSU10]), along with 2^{20} memory to accommodate the diamond.

2. Select an untested set of predefined values for $c_0, \cdots, c_7$ and initialize table L to empty.

3. Feed the 2^{32} possible inputs into the forward chunk and compute to $\#\mathrm{MC}^1_{\{1,3,4,6,9,11,12,14\}}$ with the knowledge of prefixed gray cells in target and $\#\mathrm{SB}^2_{\{0,5,10,15\}}$. For the first column of $\#\mathrm{MC}^1, \#\mathrm{SB}^2$, we have Equation 20. It's evident that the left side of the two equations can be calculated independently from the forward chunks, while the right side of the equations can be computed from the backward chunks. As a result, we store the eight bytes of left side as indexes into L, with the stored values representing $\#\mathrm{SR}^3_{\{0,1,2,3\}}$.

4. Feed the 2^{32} possible inputs into the backward chunk and compute to $\#\text{SB}^2$. Compute eight bytes of the right side of Equation 20 and search the index of L to find the full state of $\#\text{SR}^3$. Check the $2^{32+32-64} = 1$ candidates for a full match. If a full match is found, exit with the obtained preimage of the given target. Otherwise, repeat procedures from step 2.

Complexity. The time complexity of the offline phase, as indicated in step 1, is approximately $2^{76.2}$. For the online phase of the meet-in-the-middle preimage procedure, spanning from step 2 to 4, the time complexity is $2^{128-20-32} = 2^{76}$, memory complexity is 2^{32}. Consequently, the overall time complexity of the MITM Nostradamus attack can be calculated as $2^{76.2} + 2^{76}$, resulting in approximately 2^{77}. The associated memory complexity, on the other hand, stands at $0.83 \times 2^{76.161} + 2^{20} + 2^{32} \approx 2^{76}$.

5.2 7-round classical attack on AES-MMO

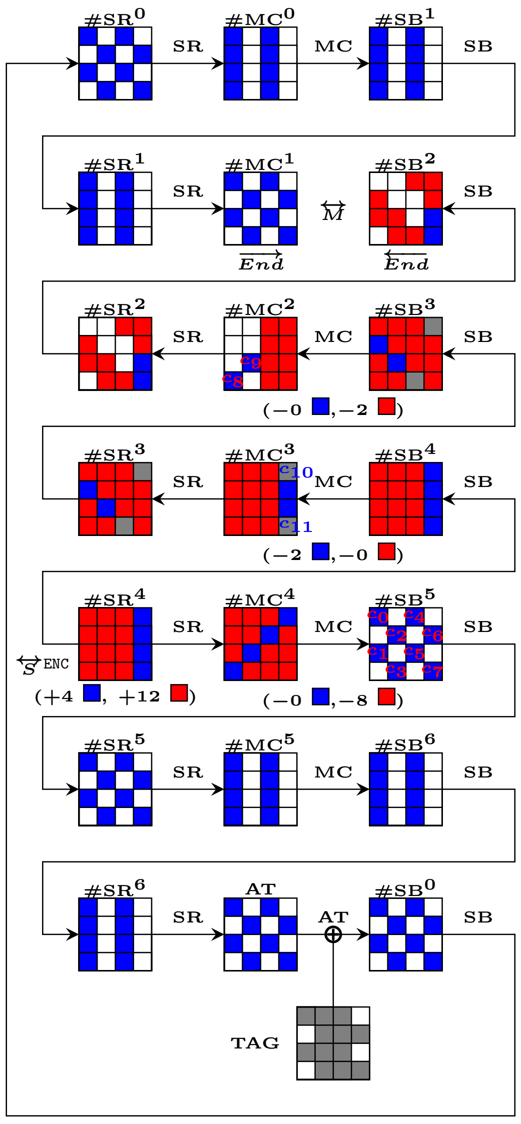


Figure 8: The 7-round attack on AES-MMO in classical setting

The configuration of the MITM attack on the 7-round AES-MMO is depicted in Figure 8. We have equivalently chosen $\#\mathrm{SR}^4$ as the initial state for both forward and backward chunks. And we have the input of the forward chunk $\mathcal{B}^{\mathtt{ENC}}$ as $\{\#\mathrm{SR}^4_{12}, \#\mathrm{SR}^4_{13}, \#\mathrm{SR}^4_{14}, \#\mathrm{SR}^4_{15}\}$ and the input of the backward chunk $\mathcal{R}^{\mathtt{ENC}} = \#\mathrm{SR}^4 \backslash \mathcal{B}^{\mathtt{ENC}}$.

Precomputation of blue initial values. A total of 2 constraints are placed on the input space of the forward chunk at the `MixColumns` operator in round 3. That is, $\#\mathrm{SB}^4_{12}$, $\#\mathrm{SB}^4_{13}$, $\#\mathrm{SB}^4_{14}$, and $\#\mathrm{SB}^4_{15}$ should have a constant impact on $\#\mathrm{MC}^3_{12}$ and $\#\mathrm{MC}^3_{15}$. As a result, the constraints can be outlined as follows, with c_{10} and c_{11} representing predefined constants:

$$\begin{aligned} 14\cdot\#\mathrm{SB}^4_{12} + 11\cdot\#\mathrm{SB}^4_{13} + 13\cdot\#\mathrm{SB}^4_{14} + 9\cdot\#\mathrm{SB}^4_{15} &= c_{10} \\ 11\cdot\#\mathrm{SB}^4_{12} + 13\cdot\#\mathrm{SB}^4_{13} + 9\cdot\#\mathrm{SB}^4_{14} + 14\cdot\#\mathrm{SB}^4_{15} &= c_{11} \end{aligned} \tag{21}$$

This leads to an input space of size $2^{2\times 8} = 2^{16}$ for the forward chunk.

Precomputation of red initial values. A total of 10 constraints are placed on the input space of the backward chunk at the `MixColumns` operator in round 2 and round 4. In round 2, $\#\mathrm{SB}^3_0$, $\#\mathrm{SB}^3_2$, $\#\mathrm{SB}^3_3$, $\#\mathrm{SB}^3_4$, $\#\mathrm{SB}^3_5$ and $\#\mathrm{SB}^3_7$ should have a constant impact on $\#\mathrm{MC}^2_3$ and $\#\mathrm{MC}^2_6$ to maintain the independence of the forward computation. As a result, the constraints can be outlined as follows, with c_8 and c_9 representing predefined constants:

$$\begin{aligned} 11\cdot\#\mathrm{SB}^3_0 + 9\cdot\#\mathrm{SB}^3_2 + 14\cdot\#\mathrm{SB}^3_3 &= c_8 \\ 13\cdot\#\mathrm{SB}^3_4 + 9\cdot\#\mathrm{SB}^3_5 + 11\cdot\#\mathrm{SB}^3_7 &= c_9 \end{aligned} \tag{22}$$

similarly in round 4, the constraints are:

$$\begin{aligned} 2\cdot\#\mathrm{MC}^4_0 + 3\cdot\#\mathrm{MC}^4_1 + 1\cdot\#\mathrm{MC}^4_2 &= c_0 \\ 1\cdot\#\mathrm{MC}^4_0 + 1\cdot\#\mathrm{MC}^4_1 + 2\cdot\#\mathrm{MC}^4_2 &= c_1 \\ 1\cdot\#\mathrm{MC}^4_4 + 2\cdot\#\mathrm{MC}^4_5 + 1\cdot\#\mathrm{MC}^4_7 &= c_2 \\ 3\cdot\#\mathrm{MC}^4_4 + 1\cdot\#\mathrm{MC}^4_5 + 2\cdot\#\mathrm{MC}^4_7 &= c_3 \\ 2\cdot\#\mathrm{MC}^4_8 + 1\cdot\#\mathrm{MC}^4_{10} + 1\cdot\#\mathrm{MC}^4_{11} &= c_4 \\ 1\cdot\#\mathrm{MC}^4_8 + 2\cdot\#\mathrm{MC}^4_{10} + 3\cdot\#\mathrm{MC}^4_{11} &= c_5 \\ 2\cdot\#\mathrm{MC}^4_{13} + 3\cdot\#\mathrm{MC}^4_{14} + 1\cdot\#\mathrm{MC}^4_{15} &= c_6 \\ 1\cdot\#\mathrm{MC}^4_{13} + 1\cdot\#\mathrm{MC}^4_{14} + 2\cdot\#\mathrm{MC}^4_{15} &= c_7 \end{aligned} \tag{23}$$

This would lead to an input space of size $2^{2\times 8} = 2^{16}$ for the backward chunk.

Match between the matching states. The forward and backward computations conclude at $\#\mathrm{MC}^1$ and $\#\mathrm{SB}^2$, and the matching process will be conducted in column 3. We have obtained $\#\mathrm{MC}^1_{13}$, $\#\mathrm{MC}^1_{15}$, $\#\mathrm{SB}^2_{14}$, and $\#\mathrm{SB}^2_{15}$ during the forward computation, and $\#\mathrm{SB}^2_{12}$, $\#\mathrm{SB}^2_{13}$ during the backward computation. If a partial match is present for column 3, it should adhere to the following:

$$\begin{cases} \#\mathrm{MC}^1_{13} = 9\cdot\#\mathrm{SB}^2_{12} + 14\cdot\#\mathrm{SB}^2_{13} + 11\cdot\#\mathrm{SB}^2_{14} + 13\cdot\#\mathrm{SB}^2_{15} \\ \#\mathrm{MC}^1_{15} = 11\cdot\#\mathrm{SB}^2_{12} + 13\cdot\#\mathrm{SB}^2_{13} + 9\cdot\#\mathrm{SB}^2_{14} + 14\cdot\#\mathrm{SB}^2_{15} \end{cases} \tag{24}$$

which leads to

$$\begin{cases} \#\mathrm{MC}^1_{13} - 11\cdot\#\mathrm{SB}^2_{14} - 13\cdot\#\mathrm{SB}^2_{15} = 9\cdot\#\mathrm{SB}^2_{12} + 14\cdot\#\mathrm{SB}^2_{13} \\ \#\mathrm{MC}^1_{15} - 9\cdot\#\mathrm{SB}^2_{14} - 14\cdot\#\mathrm{SB}^2_{15} = 11\cdot\#\mathrm{SB}^2_{12} + 13\cdot\#\mathrm{SB}^2_{13} \end{cases} \tag{25}$$

This would result in a filter of $2^{8\times 2} = 2^{16}$.

Attack Procedure. The meet-in-the-middle Nostradamus attack procedure is executed in the following manner:

1. Set the values of the 96-bit gray cells and one bit of any white cell (*e.g.*, the first bit of TAG_1) in the target to zeros. Construct the diamond structure with $2^{128-96-1} = 2^{31}$ leaves which follows the configuration of the target, necessitating a time complexity of $\sqrt{31} \times 2^{(128+31)/2} = 2^{82}$ and a memory complexity of approximately 0.83×2^{82}, along with 2^{31} memory to accommodate the diamond.

2. Select an untested set of predefined values for $c_0, \cdots, c_{11}$ and initialize table L to empty.

3. Feed the 2^{16} possible inputs into the forward chunk and compute to $\#\mathrm{MC}^1_{\{13,15\}}$ with the knowledge of prefixed gray cells in target and $\#\mathrm{SB}^2_{\{14,15\}}$. From Equation 25, it is evident that the left side of the two equations can be calculated independently from the forward chunks, while the right side of the equations can be computed from the backward chunks. As a result, we store the two bytes of left side as indexes into L, with the stored values representing $\#\mathrm{SR}^4_{\{12,13,14,15\}}$.

4. Feed the 2^{16} possible inputs into the backward chunk and compute to $\#\mathrm{SB}^2$. Compute two bytes of the right side of Equation 25 and search the index of L to find the full state of $\#\mathrm{SR}^4$. Check the $2^{16+16-16} = 2^{16}$ candidates for a full match. If a full match is found, exit with the obtained preimage of the given target. Otherwise, repeat procedures from step 2.

Complexity. The time complexity of the offline phase, as indicated in step 1, is approximately 2^{82}. For the online phase of the meet-in-the-middle preimage procedure, spanning from step 2 to 4, the time complexity is $2^{128-31-16} = 2^{81}$, memory complexity is 2^{16}. Consequently, the overall time complexity of the MITM Nostradamus attack can be calculated as $2^{82} + 2^{81}$, resulting in approximately 2^{83}. The associated memory complexity, on the other hand, stands at $0.83 \times 2^{82} + 2^{31} + 2^{16} \approx 2^{82}$.

5.3 Improved 7-round attack on `AES-MMO` in quantum setting

The configuration of the quantum MITM attack on 7-round `AES-MMO` is shown in Figure 9. The size of the offline diamond structure is chosen to be 2^{14}, which results in a time complexity of the offline phase amounting to $\sqrt[3]{14} \cdot 2^{(128+2\cdot 14)/3} = 2^{53.3}$, along with a requirement for $14^{-\frac{2}{3}} \cdot 2^{(128+2\cdot 14)/3} = 2^{49.5}$ QRACM qRAM and 2^{14} classical memory to store the diamond. The size of N_+ and N_- are $2^{(4-1-1)\times 8} = 2^{16}$ and $2^{(12-8-3)\times 8} = 2^8$. Given the size of M as 2^8, when applying the quantum formula, the time complexity of the MITM preimage attack becomes $2^{\frac{128-14-8}{2}} = 2^{53}$. Consequently, the combined complexity of the quantum Nostradamus attack is calculated as $2^{53.3} + 2^{53}$, resulting in a total of $2^{54.1}$. The memory complexity involves $2^{49.5}$ QRACM qRAM, 2^{14} classical memory, and 2^8 QRAQM qRAM for the quantum MITM approach.

Low-qRAM variant. We can use the algorithm presented in [DLPZ23] to suit the low qRAM scenario. Constructing a diamond structure of size $2^k = 2^6$ entails a time complexity of $k^{1/5} \cdot 2^{(2n+4k)/5} = 2^{56.5}$, classical memory usage of $k^{3/5} \cdot 2^{(n+2k)/5} = 2^{30}$, and an qRAM requirement of $O(n)$. When combined with the MITM attack with a time complexity of $2^{(128-8-6)/2} = 2^{57}$ and 2^8 QRAQM qRAM, the resulting overall time complexity becomes $2^{57} + 2^{56.5} = 2^{58}$, while classical memory remains at $2^{30} + 2^6 \approx 2^{30}$, and qRAM at 2^8 QRAQM.

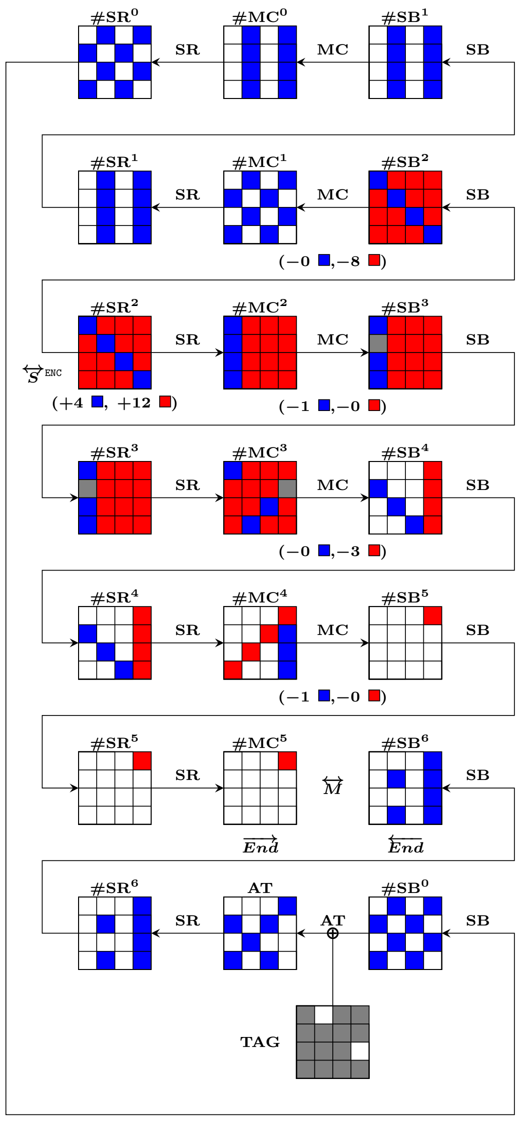


Figure 9: The 7-round quantum attack on `AES-MMO`

Remark 3. When considering a quantum attack on the 7-round `AES-MMO`, it's important to note that achieving a better time complexity than what is presented in [ZSWH23] is not feasible. While we can introduce white cells to expand the search space, we observe that the degree of freedom is mainly constrained by factors such as blue cells, red cells, matching bytes, and the difference $|d_B - d_R|$, which cannot exceed 1. In the quantum setting, the online phase of the Nostradamus attack is determined by $2^{(n-k-\min(|d_{\mathcal{B}}-d_{\mathcal{R}}|,d_{\mathcal{B}},d_{\mathcal{R}},d_{\mathcal{M}}))/2}$. As a result, the time complexity of online phase is $2^{(128-k-8)/2} = 2^{60-k/2}$. To achieve a balance with offline diamond construction phase whose time complexity is $\sqrt[3]{k} \cdot 2^{(128+2k)/3}$, the optimal value for k is determined to be 14. Therefore, as long as the number of white bytes exceeds 2, setting k to 14 ensures the optimal total time complexity. Therefore, regardless of the number of degrees of freedom obtained from the TAG element, the ultimate complexity for achieving balance must be precisely 14-bit degrees of freedom. This results in a balanced value of $2^{54.1}$.

5.4 6-round classical attack on Whirlpool

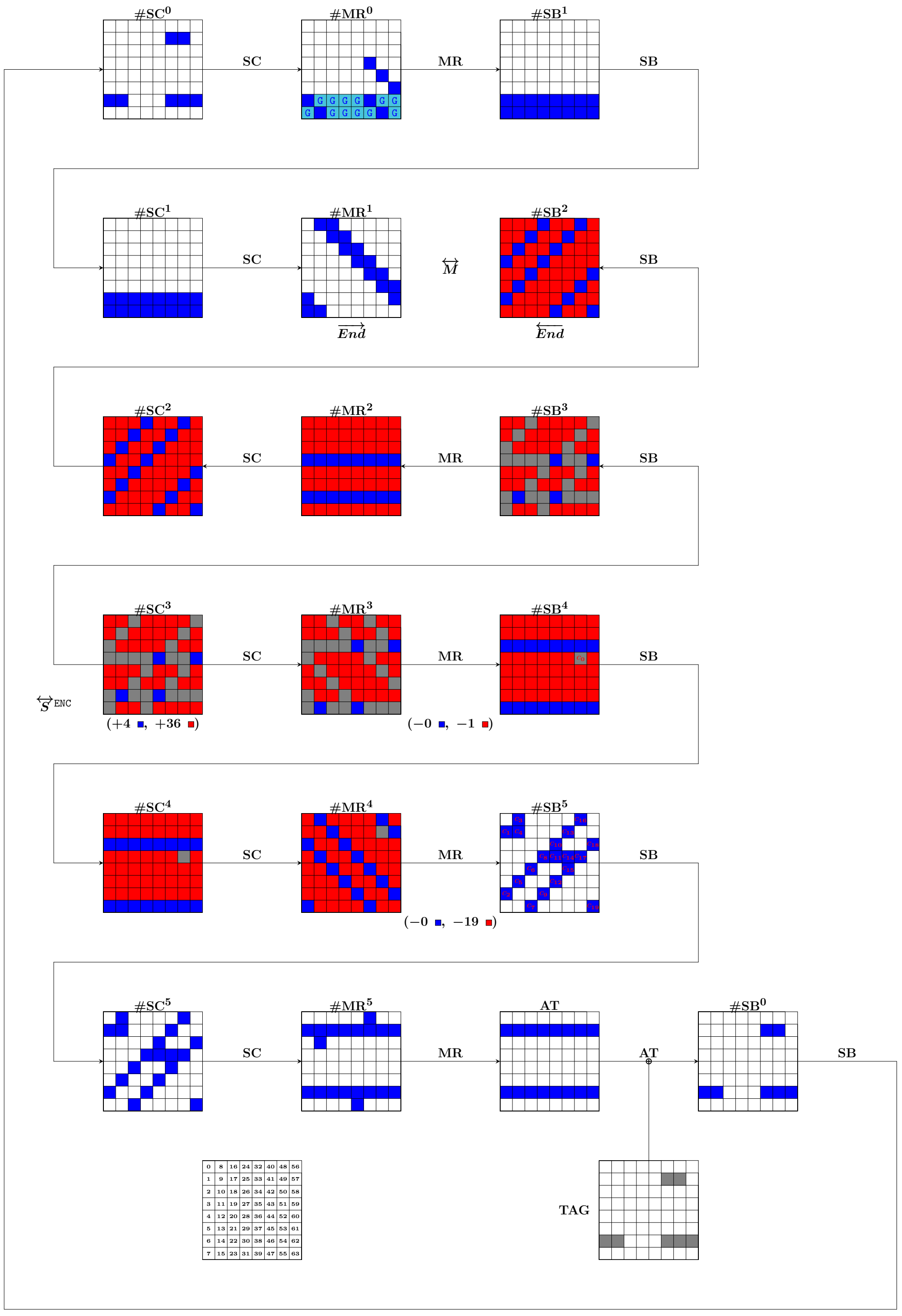


Figure 10: The 6-round classical attack on Whirlpool

The configuration of the MITM attack on 6-round `Whirlpool` is shown in Figure 10. We have equivalently chosen $\#\mathrm{SC}^3$ as the initial state for both forward and backward chunks. And we have the input of the forward chunk $\mathcal{B}^{\texttt{ENC}}$ as $\{\#\mathrm{SC}^3_{14}, \#\mathrm{SC}^3_{35}, \#\mathrm{SC}^3_{38}, \#\mathrm{SC}^3_{59}\}$ and the input of the backward chunk $\mathcal{R}^{\texttt{ENC}}$ as 36 red cells in $\#\mathrm{SC}^3$.

Precomputation of blue initial and to-be-guessed values. Given that there are no restrictions applied to the forward chunk, we have the freedom to select the values of $\#\mathrm{SC}^3_{14}$, $\#\mathrm{SC}^3_{35}$, $\#\mathrm{SC}^3_{38}$, $\#\mathrm{SC}^3_{59}$ and the values of guessed bytes of $\#\mathrm{MR}^0_{\{7,14,22,23,30,31,38,39,47,54,62,63\}}$, then propagate them to the corresponding states $\#\mathrm{MR}^1$ and $\#\mathrm{SB}^2$. The input space of blue bytes has a size of $2^{16\times 8} = 2^{128}$.

Precomputation of red initial values. A total of 20 constraints are placed on the input space of the backward chunk at the `MixColumns` operator in round 3 and round 4. In round 3, $\#\mathrm{MR}^3_{11}$, $\#\mathrm{MR}^3_{19}$, $\#\mathrm{MR}^3_{27}$, $\#\mathrm{MR}^3_{35}$, $\#\mathrm{MR}^3_{51}$ and $\#\mathrm{MR}^3_{59}$ should have a constant impact on $\#\mathrm{SB}^4_{51}$. The constraints can be outlined as follows, with c_0 representing predefined constants:

$$5\cdot\#\mathrm{MR}^3_{11} + 8\cdot\#\mathrm{MR}^3_{19} + 1\cdot\#\mathrm{MR}^3_{27} + 4\cdot\#\mathrm{MR}^3_{35} + 1\cdot\#\mathrm{MR}^3_{51} + 9\cdot\#\mathrm{MR}^3_{59} = c_0 \quad (26)$$

While in round 4, the constraints are similarly put on $c_1,\ldots,c_{19}$. This would lead to an input space of size $2^{(36-20)\times 8} = 2^{128}$ for the backward chunk.

Match between the matching states. The forward and backward computations conclude at $\#\mathrm{MR}^1$ and $\#\mathrm{SB}^2$, and the matching process will be conducted row by row. For example, in row 0, we have obtained $\#\mathrm{MR}^1_8$, $\#\mathrm{MR}^1_{16}$, and $\#\mathrm{SB}^2_{24}$, $\#\mathrm{SB}^2_{48}$ during the forward computation, and $\#\mathrm{SB}^2_0$, $\#\mathrm{SB}^2_8$, $\#\mathrm{SB}^2_{16}$, $\#\mathrm{SB}^2_{32}$, $\#\mathrm{SB}^2_{40}$, and $\#\mathrm{SB}^2_{56}$ during the backward computation. If a partial match is present for column 0, it should adhere to the following:

$$\begin{cases} \#\mathrm{MR}^1_8 = 175\cdot\#\mathrm{SB}^2_0 + 4\cdot\#\mathrm{SB}^2_8 + 62\cdot\#\mathrm{SB}^2_{16} + 203\cdot\#\mathrm{SB}^2_{24} + 194\cdot\#\mathrm{SB}^2_{32} \\ \qquad +194\cdot\#\mathrm{SB}^2_{40} + 164\cdot\#\mathrm{SB}^2_{48} + 14\cdot\#\mathrm{SB}^2_{56} \\ \#\mathrm{MR}^1_{16} = 14\cdot\#\mathrm{SB}^2_0 + 175\cdot\#\mathrm{SB}^2_8 + 4\cdot\#\mathrm{SB}^2_{16} + 62\cdot\#\mathrm{SB}^2_{24} + 203\cdot\#\mathrm{SB}^2_{32} \\ \qquad +194\cdot\#\mathrm{SB}^2_{40} + 194\cdot\#\mathrm{SB}^2_{48} + 164\cdot\#\mathrm{SB}^2_{56} \end{cases} \quad (27)$$

which leads to

$$\begin{cases} \#\mathrm{MR}^1_8 - 203\cdot\#\mathrm{SB}^2_{24} - 164\cdot\#\mathrm{SB}^2_{48} = 175\cdot\#\mathrm{SB}^2_0 + 4\cdot\#\mathrm{SB}^2_8 + 62\cdot\#\mathrm{SB}^2_{16} \\ \qquad +194\cdot\#\mathrm{SB}^2_{32} + 194\cdot\#\mathrm{SB}^2_{40} + 14\cdot\#\mathrm{SB}^2_{56} \\ \#\mathrm{MR}^1_{16} - 62\cdot\#\mathrm{SB}^2_{24} - 194\cdot\#\mathrm{SB}^2_{48} = 14\cdot\#\mathrm{SB}^2_0 + 175\cdot\#\mathrm{SB}^2_8 + 4\cdot\#\mathrm{SB}^2_{16} \\ \qquad +203\cdot\#\mathrm{SB}^2_{32} + 194\cdot\#\mathrm{SB}^2_{40} + 164\cdot\#\mathrm{SB}^2_{56} \end{cases} \quad (28)$$

where $\mathrm{cir}(4,175,14,164,194,194,203,62)$ is the inverse matrix of `Whirlpool` linear diffusion layer $\mathrm{cir}(1,1,4,1,8,5,2,9)$.

Likewise, for rows $1,\ldots,7$, an additional 14 equations would be necessary to maintain the partial match. This would result in a filter of $2^{16\times 8} = 2^{128}$.

Attack Procedure. The meet-in-the-middle Nostradamus attack procedure is executed in the following manner:

1. Set the values of the 56-bit gray cells and 308 bits out of 456 white cells (*e.g.*, 38 white bytes of the top 5 rows and 4-bit of TAG_5) in the target to zeros. Construct the diamond structure with $2^{456-308} = 2^{148}$ leaves which follows the configuration of the target, necessitating a time complexity of $\sqrt{148}\times 2^{(512+148)/2} = 2^{333.6}$ and a memory complexity of approximately $0.83\times 2^{333.6}$, along with 2^{148} memory to accommodate the diamond.

2. Select an untested set of predefined values for $c_0, \cdots, c_{19}$ and the 192-bit gray cells in the $\#SC^3$, initialize table L to empty.

3. Feed the 2^{128} possible inputs into the forward chunk and compute to blue cells of $\#MR^1$ and $\#SB^2$ with the knowledge of prefixed gray cells in target and guessed cells in $\#MR^0$. For the first row of $\#MR^1, \#SB^2$, we have Equation 28. It's evident that the left side of the two equations can be calculated independently from the forward chunks, while the right side of the equations can be computed from the backward chunks. As a result, we store the 16 bytes of left side as indexes into L, with the stored values representing $\#SC^3_{14,35,38,59}$ and $\#MR^0_{\{7,14,22,23,30,31,38,39,47,54,62,63\}}$.

4. Feed the 2^{128} possible inputs into the backward chunk and compute to $\#SB^2$. Compute 16 bytes of the right side of Equation 28 and search the index of L to find the full state of $\#SC^3$. Check the $2^{128+128-128} = 2^{128}$ candidates for the values of guessed bytes of $\#MR^0$, the probability of making a correct guess is $2^{-12\times 8} = 2^{-96}$, check the remaining $2^{128-96} = 2^{32}$ candidates for a full match. If a full match is found, exit with the obtained preimage of the given target. Otherwise, repeat procedures from step 2.

Complexity. The time complexity of the offline phase, as indicated in step 1, is approximately $2^{333.6}$. For the online phase of the meet-in-the-middle preimage procedure, spanning from step 2 to 4, the time complexity is $2^{512-148-32-128} \cdot (2^{128}+2^{128}+2^{128}) = 2^{332}$, memory complexity is 2^{128}. Consequently, the overall time complexity of the MITM Nostradamus attack can be calculated as $2^{333.6} + 2^{332}$, resulting in approximately 2^{334}. The associated memory complexity, on the other hand, is $0.83 \times 2^{333.6} + 2^{148} + 2^{128} \approx 2^{333}$.

5.5 Improved 6-round attack on Whirlpool in quantum setting

The configuration of the quantum MITM attack on 6-round `Whirlpool` is shown in Figure 11. The size of diamond is chosen to be 2^{61}, time complexity is $\sqrt[3]{61} \cdot 2^{(512+2\times 61)/3} = 2^{213.3}$ and memory complexity is $61^{-\frac{2}{3}} \cdot 2^{(512+2\times 61)/3} = 2^{207.4}$ QRACM qRAM. The size of N_+ and N_- are $2^{(48-14-12-16)\times 8} = 2^{48}$ and $2^{(16-1-12)\times 8} = 2^{24}$. The size of M is 2^{24}, the time complexity of the MITM attack is $2^{(512-61-24)/2} = 2^{213.5}$, then the overall time complexity of the quantum Nostradamus attack is $2^{213.3} + 2^{213.5} = 2^{214}$. The memory complexity involves $2^{207.4}$ QRACM qRAM, 2^{61} classical memory, and 2^{24} QRAQM qRAM for the quantum MITM approach.

Low-qRAM variant. We can modify the algorithm presented in [DLPZ23] to suit the low qRAM scenario. Constructing a diamond structure of size $2^k = 2^{29}$ entails a time complexity of $k^{1/5} \cdot 2^{(2n+4k)/5} = 2^{229}$, classical memory usage of $k^{3/5} \cdot 2^{(n+2k)/5} = 2^{117}$, and an qRAM requirement of $O(n)$. When combined with the MITM attack with a time complexity of $2^{(512-24-29)/2} = 2^{229.5}$ and 2^{24} QRAQM qRAM, the resulting overall time complexity becomes $2^{229}+2^{229.5} = 2^{230}$, while classical memory remains at $2^{117}+2^{29} \approx 2^{117}$, and qRAM at 2^{24}.

6 Conclusion

In this paper, we have revised the multi-target technique integrated into the meet-in-the-middle automatic search framework. This change results in a reduction of time complexity during the online linking phase, consequently decreasing the overall attack time complexity in both classical and quantum settings. Through adaptive revisions of the MILP model used for searching meet-in-the-middle trails, we identify more efficient trails that not only

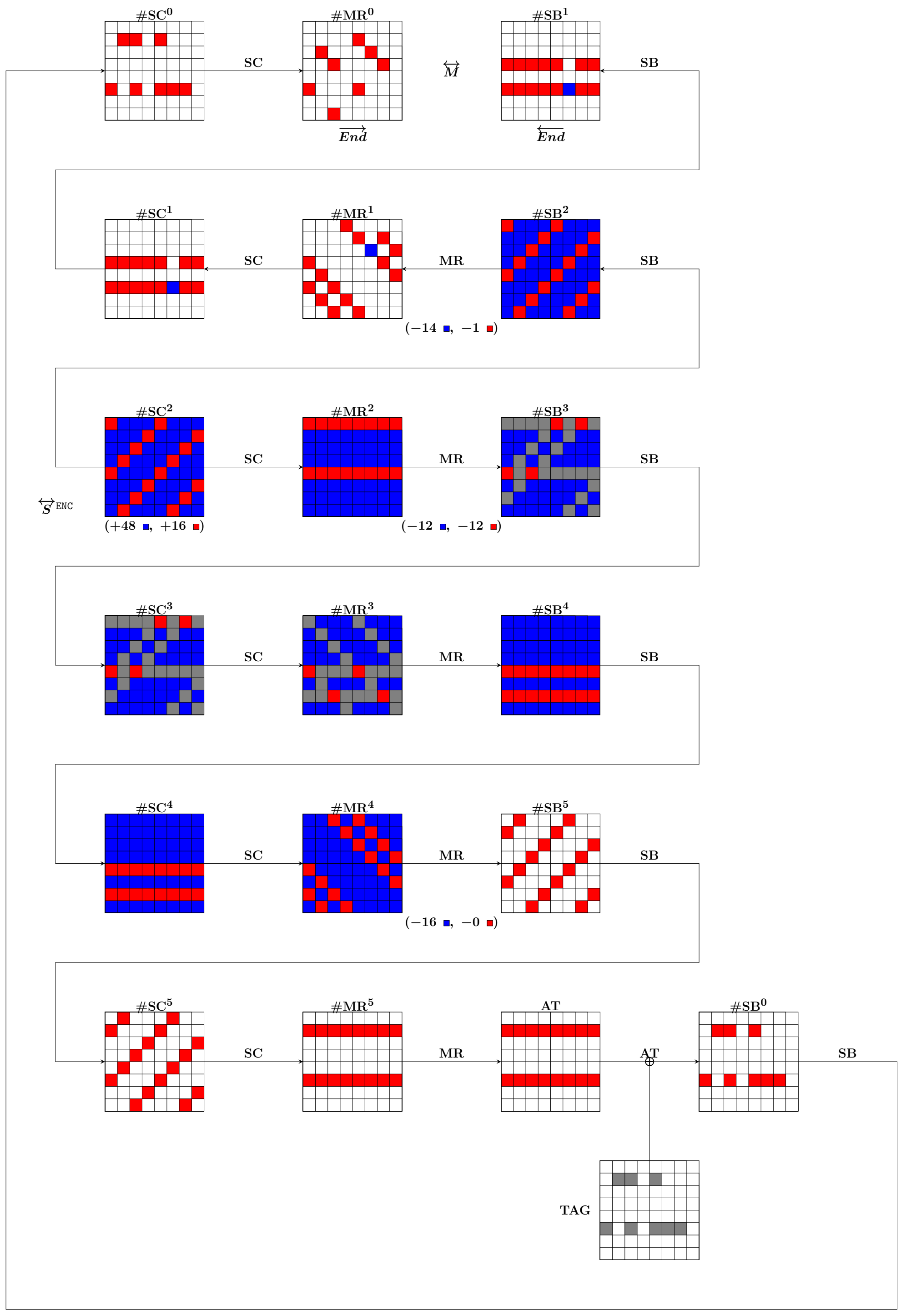


Figure 11: The 6-round quantum attack on Whirlpool

enhance time complexity but also enable the realization of additional rounds within the classical setting. We apply our attack in various scenarios: conducting a classical MITM Nostradamus attack on a 7-round `AES-MMO` and a 6-round `Whirlpool`, and achieving an enhanced quantum MITM Nostradamus attack on both a 7-round `AES-MMO` and a 6-round

`Whirlpool`.

Acknowledgments

We would like to thank Eik List, Shiyao Chen and the anonymous reviewers of ToSC for their valuable comments and suggestions. This research is partially supported by the National Key R&D Program of China (Grants No. 2022YFB2701900) and Ministry of Education in Singapore under Grants RG93/23. Xiaoyang Dong is supported by National Key R&D Program of China (2022YFB2702804, 2018YFA0704701), the Natural Science Foundation of China (62272257, 62302250, 62202017), Shandong Key Research and Development Program (2020ZLYS09), the Major Scientific and Technological Innovation Project of Shandong, China (2019JZZY010133), the Major Program of Guangdong Basic and Applied Research (2019B030302008, 2022A1515140090), Key Research Project of Zhejiang Province, China (2023C01025).

References

[ABDK09] Elena Andreeva, Charles Bouillaguet, Orr Dunkelman, and John Kelsey. Herding, second preimage and trojan message attacks beyond merkle-damgård. In *Selected Areas in Cryptography: 16th Annual International Workshop, SAC 2009, Calgary, Alberta, Canada, August 13-14, 2009, Revised Selected Papers 16*, pages 393–414. Springer, 2009.

[AGJ+15] Srinivasan Arunachalam, Vlad Gheorghiu, Tomas Jochym-O'Connor, Michele Mosca, and Priyaa Varshinee Srinivasan. On the robustness of bucket brigade quantum RAM. In *TQC 2015, May 20-22, 2015, Brussels, Belgium*, pages 226–244, 2015.

[BBG+09] Ryad Benadjila, Olivier Billet, Henri Gilbert, Gilles Macario-Rat, Thomas Peyrin, Matt Robshaw, and Yannick Seurin. Sha-3 proposal: Echo. *Submission to NIST (updated)*, page 113, 2009.

[BDG+19] Zhenzhen Bao, Lin Ding, Jian Guo, Haoyang Wang, and Wenying Zhang. Improved meet-in-the-middle preimage attacks against AES hashing modes. *IACR Trans. Symm. Cryptol.*, 2019(4):318–347, 2019.

[BDG+21] Zhenzhen Bao, Xiaoyang Dong, Jian Guo, Zheng Li, Danping Shi, Siwei Sun, and Xiaoyun Wang. Automatic search of meet-in-the-middle preimage attacks on AES-like hashing. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 771–804. Springer, Heidelberg, October 2021.

[BFH22] Barbara Jiabao Benedikt, Marc Fischlin, and Moritz Huppert. Nostradamus goes quantum. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part III*, volume 13793 of *LNCS*, pages 583–613. Springer, Heidelberg, December 2022.

[BGLP22] Zhenzhen Bao, Jian Guo, Shun Li, and Phuong Pham. Evaluating the security of merkle-damgård hash functions and combiners in quantum settings. In Xingliang Yuan, Guangdong Bai, Cristina Alcaraz, and Suryadipta Majumdar, editors, *Network and System Security - 16th International Conference, NSS 2022, Denarau Island, Fiji, December 9-12, 2022, Proceedings*, volume 13787 of *Lecture Notes in Computer Science*, pages 687–711. Springer, 2022.

[BGST22] Zhenzhen Bao, Jian Guo, Danping Shi, and Yi Tu. Superposition meet-in-the-middle attacks: Updates on fundamental security of AES-like hashing. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part I*, volume 13507 of *LNCS*, pages 64–93. Springer, Heidelberg, August 2022.

[BHMT02] Gilles Brassard, Peter Hoyer, Michele Mosca, and Alain Tapp. Quantum amplitude amplification and estimation. *Contemporary Mathematics*, 305:53–74, 2002.

[BHN$^+$19] Xavier Bonnetain, Akinori Hosoyamada, María Naya-Plasencia, Yu Sasaki, and André Schrottenloher. Quantum attacks without superposition queries: The offline Simon's algorithm. In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019, Part I*, volume 11921 of *LNCS*, pages 552–583. Springer, Heidelberg, December 2019.

[BLNS21] Xavier Bonnetain, Gaëtan Leurent, María Naya-Plasencia, and André Schrottenloher. Quantum linearization attacks. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part I*, volume 13090 of *LNCS*, pages 422–452. Springer, Heidelberg, December 2021.

[BNS19a] Xavier Bonnetain, María Naya-Plasencia, and André Schrottenloher. On quantum slide attacks. In *SAC 2019, Waterloo, ON, Canada, August 12-16, 2019*, pages 492–519, 2019.

[BNS19b] Xavier Bonnetain, María Naya-Plasencia, and André Schrottenloher. Quantum security analysis of AES. *IACR Trans. Symmetric Cryptol.*, 2019(2):55–93, 2019.

[BR$^+$00] PSLM Barreto, Vincent Rijmen, et al. The whirlpool hashing function. In *First open NESSIE Workshop, Leuven, Belgium*, volume 13, page 14. Citeseer, 2000.

[BSU10] Simon R. Blackburn, Douglas R. Stinson, and Jalaj Upadhyay. On the complexity of the herding attack and some related attacks on hash functions. Cryptology ePrint Archive, Report 2010/030, 2010. https://eprint.iacr.org/2010/030.

[CNS17] André Chailloux, María Naya-Plasencia, and André Schrottenloher. An efficient quantum collision search algorithm and implications on symmetric cryptography. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part II*, volume 10625 of *LNCS*, pages 211–240. Springer, Heidelberg, December 2017.

[Dam90] Ivan Damgård. A design principle for hash functions. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 416–427. Springer, Heidelberg, August 1990.

[DDW20] Xiaoyang Dong, Bingyou Dong, and Xiaoyun Wang. Quantum attacks on some Feistel block ciphers. *Des. Codes Cryptogr.*, 88(6):1179–1203, 2020.

[DHS$^+$21] Xiaoyang Dong, Jialiang Hua, Siwei Sun, Zheng Li, Xiaoyun Wang, and Lei Hu. Meet-in-the-middle attacks revisited: Key-recovery, collision, and preimage attacks. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part III*, volume 12827 of *LNCS*, pages 278–308, Virtual Event, August 2021. Springer, Heidelberg.

[DLPZ23] Xiaoyang Dong, Shun Li, Phuong Pham, and Guoyan Zhang. Quantum attacks on hash constructions with low quantum random access memory. Cryptology ePrint Archive, Paper 2023/1286, 2023. https://eprint.iacr.org/2023/1286.

[DSS$^+$20] Xiaoyang Dong, Siwei Sun, Danping Shi, Fei Gao, Xiaoyun Wang, and Lei Hu. Quantum collision attacks on AES-like hashing with low quantum random access memories. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part II*, volume 12492 of *LNCS*, pages 727–757. Springer, Heidelberg, December 2020.

[GKM$^+$09] Praveen Gauravaram, Lars R Knudsen, Krystian Matusiewicz, Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S Thomsen. Grøstl-a sha-3 candidate. In *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2009.

[GLM08] Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. Architectures for a quantum random access memory. *Physical Review A*, 78(5):052310, 2008.

[GLRW10] Jian Guo, San Ling, Christian Rechberger, and Huaxiong Wang. Advanced meet-in-the-middle preimage attacks: First results on full Tiger, and improved results on MD4 and SHA-2. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 56–75. Springer, Heidelberg, December 2010.

[Hos22] Akinori Hosoyamada. Quantum speed-up for multidimensional (zero correlation) linear and integral distinguishers. *Cryptology ePrint Archive*, 2022.

[HS] Akinori Hosoyamada and Yu Sasaki. Cryptanalysis against symmetric-key schemes with online classical queries and offline quantum computations. In *CT-RSA 2018, San Francisco, CA, USA, April 16-20, 2018, Proceedings*, pages 198–218.

[HS18] Akinori Hosoyamada and Yu Sasaki. Quantum Demiric-Selçuk meet-in-the-middle attacks: Applications to 6-round generic Feistel constructions. In *SCN 2018, Amalfi, Italy, September 5-7, 2018*, pages 386–403, 2018.

[HS20] Akinori Hosoyamada and Yu Sasaki. Finding hash collisions with quantum computers by using differential trails with smaller probability than birthday bound. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part II*, volume 12106 of *LNCS*, pages 249–279. Springer, Heidelberg, May 2020.

[HS21] Akinori Hosoyamada and Yu Sasaki. Quantum collision attacks on reduced SHA-256 and SHA-512. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 616–646, Virtual Event, August 2021. Springer, Heidelberg.

[HSX17] Akinori Hosoyamada, Yu Sasaki, and Keita Xagawa. Quantum multicollision-finding algorithm. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part II*, volume 10625 of *LNCS*, pages 179–210. Springer, Heidelberg, December 2017.

[KK06] John Kelsey and Tadayoshi Kohno. Herding hash functions and the Nostradamus attack. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 183–200. Springer, Heidelberg, May / June 2006.

[KLLN16a] Marc Kaplan, Gaëtan Leurent, Anthony Leverrier, and María Naya-Plasencia. Breaking symmetric cryptosystems using quantum period finding. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part II*, volume 9815 of *LNCS*, pages 207–237. Springer, Heidelberg, August 2016.

[KLLN16b] Marc Kaplan, Gaëtan Leurent, Anthony Leverrier, and María Naya-Plasencia. Quantum differential and linear cryptanalysis. *IACR Trans. Symmetric Cryptol.*, 2016(1):71–94, 2016.

[KLMR16] Stefan Kölbl, Martin M. Lauridsen, Florian Mendel, and Christian Rechberger. Haraka - efficient short-input hashing for post-quantum applications. Cryptology ePrint Archive, Report 2016/098, 2016. https://eprint.iacr.org/2016/098.

[KM10] Hidenori Kuwakado and Masakatu Morii. Quantum distinguisher between the 3-round Feistel cipher and the random permutation. In *ISIT 2010, June 13-18, 2010, Austin, Texas, USA, Proceedings*, pages 2682–2685, 2010.

[KM12] Hidenori Kuwakado and Masakatu Morii. Security on the quantum-type Even-Mansour cipher. In *ISITA 2012, Honolulu, HI, USA, October 28-31, 2012*, pages 312–316, 2012.

[KRT07] Lars R. Knudsen, Christian Rechberger, and Søren S. Thomsen. The Grindahl hash functions. In Alex Biryukov, editor, *FSE 2007*, volume 4593 of *LNCS*, pages 39–57. Springer, Heidelberg, March 2007.

[LM17] Gregor Leander and Alexander May. Grover meets simon - quantumly attacking the FX-construction. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part II*, volume 10625 of *LNCS*, pages 161–178. Springer, Heidelberg, December 2017.

[Mer90] Ralph C. Merkle. One way hash functions and DES. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 428–446. Springer, Heidelberg, August 1990.

[NS20] María Naya-Plasencia and André Schrottenloher. Optimal merging in quantum k-xor and k-xor-sum algorithms. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part II*, volume 12106 of *LNCS*, pages 311–340. Springer, Heidelberg, May 2020.

[Sas11] Yu Sasaki. Meet-in-the-middle preimage attacks on AES hashing modes and an application to Whirlpool. In Antoine Joux, editor, *FSE 2011*, volume 6733 of *LNCS*, pages 378–396. Springer, Heidelberg, February 2011.

[Sch23] André Schrottenloher. Quantum linear key-recovery attacks using the qft. *Cryptology ePrint Archive*, 2023.

[Sho94] Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994*, pages 124–134. IEEE Computer Society, 1994.

[SS22] André Schrottenloher and Marc Stevens. Simplified MITM modeling for permutations: New (quantum) attacks. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part III*, volume 13509 of *LNCS*, pages 717–747. Springer, Heidelberg, August 2022.

[ZSWH23] Zhiyu Zhang, Siwei Sun, Caibing Wang, and Lei Hu. Classical and quantum meet-in-the-middle nostradamus attacks on aes-like hashing. *IACR Transactions on Symmetric Cryptology*, pages 224–252, 2023.