

Simplified MITM Modeling for Permutations: New (Quantum) Attacks

André Schrottenloher and Marc Stevens

Cryptology Group, CWI, Amsterdam, The Netherlands
firstname.lastname@cwi.nl

Abstract. Meet-in-the-middle (MITM) is a general paradigm where internal states are computed along two independent paths ('forwards' and 'backwards') that are then matched. Over time, MITM attacks improved using more refined techniques and exploiting additional freedoms and structure, which makes it more involved to find and optimize such attacks. This has led to the use of detailed attack models for generic solvers to automatically search for improved attacks, notably a MILP model developed by Bao et al. at EUROCRYPT 2021.

In this paper, we study a simpler MILP modeling combining a greatly reduced attack representation as input to the generic solver, together with a theoretical analysis that, for any solution, proves the existence and complexity of a detailed attack. This modeling allows to find both classical and quantum attacks on a broad class of cryptographic permutations. First, PRESENT-like constructions, with the permutations from the SPONGENT hash functions: we improve the MITM step in distinguishers by up to 3 rounds. Second, AES-like designs: despite being much simpler than Bao et al.'s, our model allows to recover the best previous results. The only limitation is that we do not use degrees of freedom from the key schedule. Third, we show that the model can be extended to target more permutations, like Feistel networks. In this context we give new Guess-and-determine attacks on reduced *Simpira v2* and *SPARKLE*.

Finally, using our model, we find several new quantum preimage and pseudo-preimage attacks (e.g. *Haraka v2*, *Simpira v2* ...) targeting the same number of rounds as the classical attacks.

Keywords: MITM Attacks · Permutation-based hashing · Preimage attacks · Merging algorithms · Quantum cryptanalysis.

1 Introduction

Meet-in-the-middle is a general attack paradigm against cryptographic primitives where internal states are computed along two independent paths ('forwards' and 'backwards') that are then matched to produce a complete path solution. MITM attacks can be traced back to Diffie and Hellman's time-memory trade-off on Double-encryption [23]. Since then, they have been successfully applied over the years on block ciphers and hash functions [26,14,36,37,1,33]. Moreover,

MITM attacks have been improved using more refined techniques and exploiting additional freedoms and structure (e.g., using internal state guesses [26], splice-and-cut [1,33], bicliques [38], 3-subset MITM [14]), which also makes it more involved to find and optimize such attacks.

An important trend in cryptanalysis is the application of automatic tools to search for improved attacks. The search of an attack of a certain form is translated into a search or optimization problem, which is solved using an off-the-shelf SAT, constraint programming (CP), Mixed Integer Linear Programming (MILP) solver. Thus the difficulty of finding an attack by hand is replaced by that of finding a proper modeling of the attack search space into a corresponding search/optimization problem. This has naturally led to a bottom-up modeling including low-level attack details, such that any solution directly corresponds to an instantiation of the attack.

MITM attacks on hash functions. Hash functions are often built from a compression function, using a simple domain extender such as Merkle-Damgård [41,21]. This compression function, in turn, can be built from a block cipher E_k using one of the twelve secure PGV modes [43], usually one of the three most common: Davies-Meyer (DM), Matyas-Meyer-Oseas (MMO) and Miyaguchi-Preneel (MP). A preimage attack on the hash function can be reduced to one on the compression function.

In [44], Sasaki introduced a MITM preimage attack on AES hashing modes targeting as much as 7 rounds. This attack already integrates advanced techniques such as the *initial structure* and *matching through MixColumns*, which is reviewed later. Bao *et al.* [3] improved the attacks of [44] by making use of degrees of freedom from the key-schedule path; that is, allowing a varying chaining value instead of considering a fixed one. In [4], an MILP framework for automatic search of MITM attacks was introduced. It applies to all *AES-based* hash functions, whose internal state is defined as an array of fixed-size cells and whose operations mimic the operations of the AES block cipher. This modeling led to many improved results; in particular, the first 8-round preimage attack on a hash function using AES-128. Later on, this modeling was improved in [5] and [25]. The former introduced the technique of *guess-and-determine* in the solver, while the latter extended the search to collision attacks and key-recovery attacks against block ciphers.

Limits of Rule-based Modeling. In AES-based hash functions, internal states are represented as an array of cells corresponding to the S-Boxes. The MITM attack can entirely be specified by a certain coloring of these cells (backwards, forwards, unspecified). *Propagation rules* can then be defined, which specify the admissible coloring transitions at each stage of the cipher, while computing the parameters which give the time and memory complexities of the MITM attack. This is a bottom-up approach, as the validity of the path is enforced locally.

However, the definition of these rules is quite involved, and the follow-up works [25,5] added even more rules to capture new techniques. This increases,

in turn, the complexity of the model, which (as reported in [5]) requires more human intervention to limit the search space.

Furthermore, the rule-based modeling in [4] is limited to AES-like ciphers. These primitives have the property that the linear layer is *strongly aligned* with the S-Box layer, and all the operations can be defined at cell-level (bytes in the case of AES), instead at bit-level. Extending the rule-based modeling to other primitives was one of the main open questions in [4], which would typically require moving to a bit-level and increasing model complexity. Our goal is to develop a powerful model that is both broadly applicable and significantly simpler than rule-based models.

Quantum Preimage Attacks. It is well-known that Grover’s quantum search algorithm [30] halves the bits of preimage security that one can expect from a hash function, e.g., instead of requiring 2^{128} computations of a 128-bit hash function, Grover’s search can find a preimage in about 2^{64} evaluations of a quantum circuit for the function. However, Grover search is only a generic algorithm. There might exist dedicated *quantum attacks* that, for a given design, find a preimage in less time. Such attacks determine the security margin of a hash function in a post-quantum context, especially for hash-based signature schemes [2]. But to date, while quantum collision attacks have been significantly studied [34,35], little is known on quantum preimage attacks.

1.1 Our Contributions

Top-down modeling. In this work we do not follow the detailed bottom-up modeling where any solution directly corresponds to an instantiation of the attack. Ideally, the modeling should remain simple, and lead to feasible search times, while at the same time, cover a large space of potential attacks. Hence instead, we study a simpler *top-down* modeling paradigm in which we search for a greatly simplified attack representation excluding many details, for which we are able to prove the existence of an optimized attack instantiation and its corresponding complexity (see Lemma 2 and Theorem 1 in Section 4). This has several benefits. First, the abstract representation makes it more generically applicable to a wide set of designs. Second, it enables analysis of not only classical attacks, but quantum attacks as well with minor changes. Third, the resulting model input to the solver is significantly smaller, which typically means it can be solved faster and thus it is more practical to cover larger primitives and/or more rounds.

MITM preimage attacks. We apply this top-down modeling paradigm to MITM preimage attacks. Our representation is close to the dedicated solvers introduced in [16,22], and complementary to the bottom-up modeling developed in [4,25,5]. Instead of defining local rules for the propagation of cell coloring between cells, we consider a global view of the MITM attack capturing only which cells belong to the forward and the backward paths, and optimize the attack time complexity as a function of the cells. This view has two advantages: first, its simplicity. Second, its genericity, as it is not limited to strongly aligned designs and allows to

target a larger class than AES-based hashing. In fact, we start with applications to “PRESENT-like” permutations, and only later, rewrite AES-based primitives as “PRESENT-like”, using the Super S-Box.

Our approach is so far limited to permutations: we do not use degrees of freedom of the key-schedule. This restriction makes our tool oblivious to the most advanced attacks on hashing using AES. However, many recent hash functions, especially small-range hash functions like *Simpira* v2 [31] or *Haraka* v2 [40], or more generally, Sponge designs like SHA-3, are only based on permutations.

Our modeling also admits a generic translation of classical MITM attacks into quantum attacks. We find these attacks using our automatic tool, by a mere change in the optimization goal. In fact, the valid paths for quantum attacks correspond to classical paths *under new memory constraints*. When applicable, our quantum attacks reach the same number of rounds as the classical ones.

Outline and Results. In [Section 2](#), we recall previous results and elaborate on the definition and modeling of a MITM attack in [4,25,5]. The rest of the paper follows our new approach. We define our *cell-coloring representation*, and *merging-based MITM attacks*, in [Section 3](#). In [Section 4](#), we simplify this representation and detail our MILP modeling for classical and quantum attacks. Next, we demonstrate the versatility of our approach and obtain existing and new state-of-the-art attacks.

In [Section 5](#), we study the class of PRESENT-like permutations, which have the same operations as the block cipher PRESENT: individual S-Boxes, followed by a linear layer which exchanges bits between pairs of S-Boxes. We improve the MITM step in the distinguishers on the permutations of the SPONGENT family.

In [Section 6](#), we study the class of AES-like permutations. With the Super S-Box, AES itself becomes a small PRESENT-like cipher. We recover previous results on these permutations and give new quantum preimage attacks on reduced-round AES, *Haraka* v2 and *Grøstl* (these results are summarized in [Table 1](#)).

In [Section 7](#), we study an extended class of permutations in which the linear layer contains XORs. In particular, we study Generalized Feistel Networks and obtain generic and *practical* guess-and-determine distinguishers on GFNs, reduced-round *Simpira* permutations, and reduced-step SPARKLE permutations (summarized in [Tables 3](#) and [4](#)). The distinguishers on *Simpira* are converted into preimage attacks (see [Table 1](#)).

Our code is available at: github.com/AndreSchrottenloher/mitm-milp. We used the MILP solver of the SCIP Optimization Suite [29].

2 Preliminaries

In this section, we describe the families of PRESENT-like, AES-like and Feistel-like permutations targeted in this paper. We recall MITM problems and the rule-based framework studied in [44,3,4,25,5]. We choose to focus only on single-target *pseudo-preimage attacks*, and refer to [3] for a clear depiction of generic techniques to convert pseudo-preimage to preimage attacks.

Table 1. Our new (pseudo)-preimage attacks, with points of comparison to previous works. QRAQM = quantum-accessible quantum memory. The generic time given can be higher than the security claims of the design.

$|_n$: A partial preimage attack over n -bits. (Q): Using QRAQM.

Target	Type	Rounds	Time	Generic time	Memory	Source
AES-128	Classical	8	2^{120}	2^{128}	2^{40}	[4]
AES-128	Quantum	7	$2^{63.34}$	2^{64}	2^8 (Q)	Section 6.1
Haraka-256 v2	Classical	4.5 / 5	2^{224}	2^{256}	2^{32}	[4]
Haraka-256 v2	Quantum	4.5 / 5	$2^{115.55}$	2^{128}	2^{32} (Q)	Section 6.2
Haraka-512 v2	Classical	5.5 / 5	2^{240}	2^{256}	2^{128}	[4]
Haraka-512 v2	Classical	5.5 / 5	2^{240}	2^{256}	2^{16}	Section 6.2
Haraka-512 v2	Quantum	5.5 / 5	$2^{123.34}$	2^{128}	2^{16} (Q)	Section 6.2
Haraka-512 v2 $ _{32}$	Classical	5.5 / 5	2^{16}	2^{32}	2^{16}	Section 6.2
Haraka-512 v2 $ _{64}$	Classical	5 / 5	2^{32}	2^{64}	2^{32}	Full version [45]
SPHINCS+-Haraka	Quantum	3.5 / 5	$2^{64.65}$	$2^{85.33}$	negl.	Section 6.2
Grøstl-256 OT	Classical	6 / 10	2^{224}	2^{256}	2^{128}	[5]
Grøstl-256 OT	Quantum	6 / 10	$2^{123.56}$	2^{128}	2^{112} (Q)	Section 6.3
Grøstl-512 OT	Classical	8 / 14	2^{472}	2^{512}	2^{224}	[5]
Grøstl-512 OT	Quantum	8 / 14	$2^{255.55}$	2^{256}	2^{56} (Q)	Section 6.3
Simpira-2	Classical	5 / 15	2^{128}	2^{256}	negl.	Section 7.2
Simpira-2	Quantum	5 / 15	2^{64}	2^{128}	negl.	Section 7.2
Simpira-4	Classical	9 / 15	2^{128}	2^{256}	negl.	Section 7.2
Simpira-4	Quantum	9 / 15	2^{64}	2^{128}	negl.	Section 7.2

2.1 Families of Designs

PRESENT-like. We name this family after the block cipher PRESENT [13]. It is a Substitution-Permutation Network (SPN) with an internal state of $b = 16$ cells of 4 bits. Its round function applies in order: (1) the round key addition, (2) the PRESENT S-Box on each cell independently, and (3) the linear layer defined by the bit-permutation:

$$P(j) = \begin{cases} 4b - 1 & \text{if } j = 4b - 1; \\ (j \cdot b) \bmod 4b - 1 & \text{otherwise.} \end{cases}$$

That is, the j -th bit of the state after an S-Box layer is moved to the $P(j)$ -th bit of the state before the next key addition. In particular, each cell at a given round connects to 4 cells at the next round. Thus, PRESENT is an SPN in the strict sense that the “permutation” is a permutation of bits. In this paper, we consider the analysis of PRESENT in the known-key setting (see e.g. [10]), where the key is fixed, which turns the cipher into a permutation. The SPONGENT- π family of permutations¹, which are used in the SPONGENT hash function [11] is a generalization of the PRESENT design to larger state sizes, with b ranging from

¹ This denomination is from [9]. Previously the permutation did not have a name, or was named “SPONGENT” by metonymy.

22 to 192. By abstracting out the S-Box, other designs such as GIMLI [8] can be considered as PRESENT-like.

AES-like. The AES, designed by Daemen and Rijmen [20], is the standardized version of the candidate Rijndael [19] which was chosen in an open competition organized by the NIST. It is a block cipher with a state of 16 bytes (128 bits). The bytes are arranged in a 4×4 array, where the byte at position (i, j) is numbered $4j + i$. Each round contains the following operations in order: (1) **AddRoundKey** (ARK): the subkey is XORed to the state, (2) **SubBytes** (SB): the 8-bit S-Box is applied to each byte independently, (3) **ShiftRows** (SR): the row number i (starting from 0) is shifted by i bytes left, and (4) **MixColumns** (MC): the columns of the state are multiplied by an MDS matrix. Importantly, all these operations can be defined *at byte level* (strong alignment).

The class of *AES-like designs* studied in previous works [4] can then be defined as follows: the internal state is an array of cells (not necessarily bytes) and the round function combines ARK, SB, MC and operations that swap cells (SR, or MIX in Haraka). In general the mixing function must be MDS, though the extension in [25] does not require this. Since we are interested in permutations, the ARK layer is replaced by **AddConstant** (AC).

Feistel-like. We consider permutations based on Generalized Feistel Networks (GFNs). The state of a GFN is formed of $b \geq 2$ *branches*. We denote branches by S_i . Apart from swapping branches, the basic operation in a GFN is to apply a *round function* F on a well-chosen pair (S_i, S_j) : $(S_i, S_j) \mapsto (S_i, S_j \oplus F(S_i))$. Our main example is the **Simpira v2** [31] family of permutations, where the branches are AES states, and the round functions apply two rounds of AES. This is an instance of the *double-SP* structures defined in [15], and a case in which the F functions are permutations.

More generally, we can extend the class of GFN to *Feistel-like permutations* by allowing permutations to be applied in place on branches, and not only through round functions: $S_i \mapsto H(S_i)$. This does not make a difference from our modeling perspective. In particular, the SPARKLE family of permutations [7] adopts such a Feistel-like structure, but with non-linear permutations on the branches, and linear mixing layers. Though it is not strictly a GFN, our modeling captures it as well.

2.2 Generic Depiction of MITM Attacks

We consider the MITM attack framework as represented in **Figure 1** using the *splice-and-cut* and *initial structure* techniques. The key schedule is ignored due to our restriction to permutations, and we reason only with the internal states.

The goal of the MITM attack is to find a sequence of internal states which satisfy a *closed computational path*: there is a relation between the value before the first round and the value after the last round. In order to do so, one starts by separating the path in two *chunks* (splice-and-cut): the **backward chunk** ◀

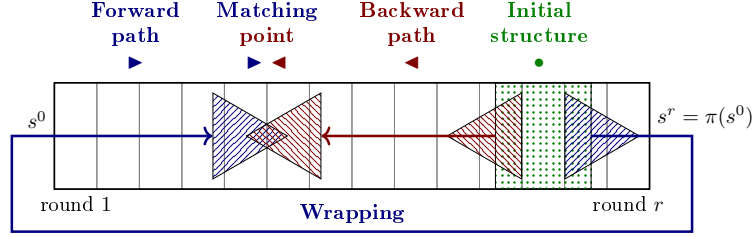


Fig. 1. MITM attack depiction with the splice-and-cut and initial structure techniques.

and the **forward chunk** \blacktriangleright . Both chunks form independent computation paths. One then finds a *partial match* between them at some round.

In addition, one usually starts at some **initial structure** \bullet which fixes some part of the internal state to constant values. The total complexity depends on (1) the amount of these global guesses, (2) the degree of freedom of both chunks, and (3) the amount of matching. All these parameters are completely determined by the definition of chunks. As an example, we detail the 7-round attack on AES of Sasaki [44] in the full version of the paper [45].

2.3 Rule-based Modeling and Limits

A MILP model for searching MITM attacks on AES-like designs has been introduced in [4] and further improved in [25,5].

Given the byte-level structure of the design, one fixes a starting round and an ending round where the matching occurs (all possibilities are enumerated). Then, each byte is ‘colored’ like in Figure 1. There are four ‘colors’ (backward, forwards, initial, unknown), which are encoded on two Boolean variables. Only the ARK (if the key-schedule is used) and MC operations change the colors. A series of *rules* is then enforced, as constraints, on the coloring transitions through these operations. For example, going through MC forwards, one “unknown” byte in input implies all bytes “unknown” in output; if all bytes are “initial” in input, then they are all “initial” in output, etc. Other constraints have to be enforced if we go backwards.

At the starting states, there are “initial degrees of freedom” which count the number of forward and backward bytes. The forward computation path, respectively backwards, consume these degrees of freedom under an enforcement of the propagation rules. There must remain enough degrees of freedom at the ending round, in order to ensure some matching.

This representation captures a large number of possible paths (including the key-schedule, contrary to this paper). However, there are several downsides. First of all, the rule-based modeling is complex, and the set of paths depends crucially on the implementation of the propagation rules. For example, the introduction of Guess-and-determine in [5] required to add more rules to take into account this additional technique. The approach so far is *bottom-up* in the sense that the

set of possible paths is defined by the local propagation rules. (In contrast, in this paper we use a *global* approach, in which the objective function is directly computed from the coloring. Advanced techniques such as Guess-and-determine are covered by design and without the need for new rules.)

Second, the above model [4] works only for AES-like designs, and extending it to bit-oriented ciphers is far from obvious, as stated in [25]. Notably, it becomes unclear how the S-Box and linear layer will interact. Our model overcomes this problem, albeit restricted to permutations.

3 Cell-coloring Representation of MITM Attacks

In this section, we define the classes of designs under study, and the class of *merging-based* MITM attacks which we are interested in. These attacks have been previously studied in [16,22] in a very generic setting in combination with a dedicated search tool. Although the search space is similar, our approach differs by using MILP instead. The basis of our representation is PRESENT-like designs. We extend it in two directions: AES-like designs on the one hand, more complex linear layers on the other hand. These are referred in this work as the “PRESENT-like setting”, the “AES-like setting” and the “extended setting”.

3.1 Cell-based Representations

Let $\pi = \pi^{r-1} \circ \dots \circ \pi^0$ be an r -round permutation. We consider the application of π to an initial state s^0 , and write s^i the state before round i . Thus s^r is the final state and we have: $\forall i \geq 0, s^{i+1} = \pi^i(s^i)$.

For now π is assumed to be a Substitution-Permutation Network (SPN). We can cut each s^i into b cells of w bits, denoted as s_j^i where $0 \leq j \leq b-1$. Each round applies individual S-Boxes S to the cells (substitution), then a linear layer between them (permutation). By abuse of notation, we also name “cell” the pair $x_j^i = (s_j^i, S(s_j^i))$. Thus cells are $2w$ -bit words, which can only take 2^w values. The linear layer of round i relates the cells x_j^i to the cells x_j^{i+1} . We have now completely unfolded the equation $s^r = \pi(s^0)$, into a system of *linear equations on the cells*. So far this view is the same as in [16,22].

PRESENT-like Setting. The archetype of a PRESENT-like design is represented on Figure 2. Here we have two rounds with 4 cells each, of 4 bits. The linear layer merely swaps bits. Thus, it can be entirely represented by pairwise linear relations between the cells. All the information necessary for finding attacks then holds in a simple directed, weighted graph $G = (N, E)$:

- a node $x \in N$ is a cell x with a *width* parameter w_x ;
- an edge $(x, x') \in E$ is a linear relation between a cell x at a given round, and a cell x' at the next round, with a *width* $w_{x,x'}$ (we use purposefully the same term as for cells).

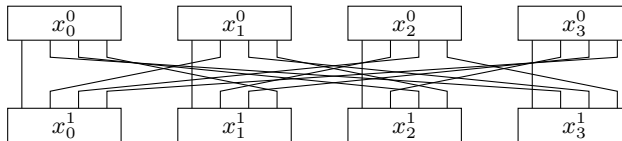


Fig. 2. Example of a 4-cell PRESENT-like design.

The width of a cell corresponds to the combined width that a set of edges needs to have to determine the cell’s value. Hence, the widths of cells and edges are relative to each other. We set the width of cells to 1, and the width of edges to a fraction (0.25 in [Figure 2](#)). It follows from the PRESENT-like structure that the combined width of incoming edges, resp., of outgoing edges, is equal to the width of the cell:

$$\forall x \in N, \quad \sum_{x'|(x,x') \in E} w_{x,x'} = \sum_{x'|(x',x) \in E} w_{x',x} = w_x . \quad (1)$$

To simplify, we make the following assumption on the S-Boxes similar to the “heuristic assumption” in Section 4.1 of [\[16\]](#). It would be true on average if the S-Boxes were drawn at random, and it is not true for fixed S-Boxes. Our final complexity estimates rely in fact on a global heuristic, rather than this local one.

Assumption 1 (S-Boxes) *Given fixed edges with a combined width $u \leq 1$, a cell x of w bits can take exactly $2^{w(1-u)}$ values.*

AES-like Setting. Our cell-based representation of AES-like designs is different from the one in previous works like [\[16,22,4\]](#). These works considered the S-Boxes as individual cells. Instead, we want to represent AES-like operations in a way that looks like a PRESENT-like design, with linear relations between pairs of cells. For this we use the *Super S-Box representation*.

In the analysis of AES, the Super S-Box consists in considering the MC operation, followed by SB, as a single, large S-Box of $4 \times 8 = 32$ bits. In our representation, the cells are the columns of a given AES-like state, as represented on [Figure 3](#) (or the rows, if MixColumns were to be replaced by MixRows). In that case, the MC operation is the one of the previous round, and the SR operation becomes an exchange of bytes between super-cells: two rounds of AES can then be represented as in [Figure 2](#). The relative widths of cells and edges are unchanged; each edge represents a byte, and each cell a column of 32 bits.

Extended Setting. In order to target even more designs, we show how to model any linear layer for which a bit of x^i is obtained by XORing several bits of x^{i-1} . This allows for example to model the permutation ASCON [\[24\]](#) (though we did not obtain interesting results on this design). This XOR operation requires the introduction of new cells:

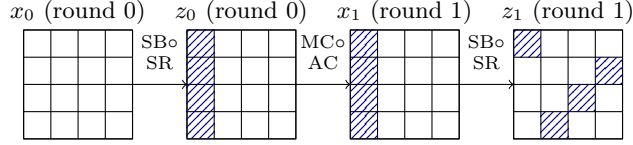


Fig. 3. 2 rounds of AES, with a single (super-)cell.

- *b-branching cells*: a cell x of width $w_x = v$, with one incoming edge and b outgoing edges of width v each;
- *b-XOR cells*: a cell x of width $w_x = bv$, with b incoming edges and one outgoing edge of width v each. The inputs are b bits, and the output is the XOR of them.

These cells allow to keep a graph structure, where the width of a cell still corresponds to a combined width of edges that allows to determine the cell's value. In *b-branching* cells, all edges have the same value, and in *b-XOR* cells, knowing b edges allows to deduce the remaining one. The difference with PRESENT-like designs is that Eq. 1 is not satisfied anymore. In order to separate successive rounds, three layers for a single round (S-Box, branching, XOR) may be needed.

3.2 Meet-in-the-middle Problems

The goal of a MITM attack is, using the cell-based representation, to find values for all cells such that a given equation system is satisfied. The starting equation system encoding $s^r = \pi(s^0)$ is trivial, where s^r can be computed from s^0 and vice-versa. By adding new linear relations between s^0 and s^r , this becomes a *closed computational path*. The relations between s^0 and s^r can also be encoded into the undirected graph of Section 3.1. We mostly consider *wrapping constraints*, where we put new edges between cells s^0 and s^r , and *input-output constraints*, where we fix some bits in the s^0 and s^r to arbitrary constants.

Problem 1 (Meet-in-the-middle problem). Consider a permutation $\pi(s^0) = s^r$. Then given either u_w bits of wrapping constraints $L(s^0, s^r) = 0$; or instead u_i bits of input constraints $L(s^0) = 0$ together with u_o bits of output constraints $L(s^r) = 0$, find a pair of states (s^0, s^r) that satisfy these constraints. (Here each L is a linear function over \mathbb{F}_2 .)

Given query access (forwards and backwards) to a random permutation, an adversary must make respectively $\mathcal{O}(2^{u_w})$ and $\mathcal{O}(2^{\min(u_i, u_o)})$ queries to solve Problem 1. These complexities are to be multiplied by the number of requested solutions. This defines the generic difficulty of the problem. Note that for solutions to exist, u_w (resp. $u_i + u_o$) cannot exceed the state size of the permutation. The number of solutions of the problem (in \log_2) can be computed by:

$$\left(\sum_{x \in N} w_x - \sum_{(x, x') \in E} w_{x, x'} \right) - u_i - u_o, \quad (2)$$

where the sum over all edges includes wrapping constraints (if applicable).

3.3 Merging-based MITM Attacks

Now that we have defined the cell-based representation, we can move on to the definition of *merging-based attacks*. This class of attacks is borrowed from [16,22]. However, while they pursue a dedicated bottom-up solver to automatically search for attacks, we follow a top-down MILP modeling approach. We focus for now on the basic PRESENT-like setting.

Reduced Lists. Let us consider a set of cells $X = (x_j^i)_{(i,j) \in IJ_X}$, i.e., nodes in the directed graph $G = (N, E)$ that represents the MITM equation system. We define the *reduced list* $\mathcal{R}[X]$ as the set of all value assignments $(v_j^i)_{(i,j) \in IJ_X}$ to X that satisfy all linear constraints between the cells of X .

E.g., we may consider in [Figure 2](#) a reduced list $\mathcal{R}[x_0^0, x_0^1]$, which contains all assignments $(s_0^0, S(s_0^0)), (s_0^1, S(s_0^1))$ such that $(S(s_0^0))|_1 = (s_0^1)|_0$ (the second bit of $S(s_0^0)$ is equal to the first bit of s_0^1). In particular, the list has size $|\mathcal{R}[x_0^0, x_0^1]| = 2^7$.

A reduced list is entirely determined by its defining set of cells. It forms the set of solutions to a subsystem of equations. Our goal can now be rephrased as follows: *Compute an element from the reduced list of all cells: $\mathcal{R}[\{x|x \in N\}]$.* Indeed, by definition, this is a solution to the MITM equation system.

Base Lists. We start with base lists: reduced lists $\mathcal{R}[\{x_j^i\}]$ of individual cells. These are simply the list of all input-outputs through the S-Box: $(s_j^i, S(s_j^i))$. In extended mode, base lists for branching and XOR cells are likewise trivial.

Merging Lists. *Merging* is the fundamental algorithmic operation to construct bigger lists. It corresponds to the “recursive combinations of solvers” considered in Section 4.2 of [16], where the “solvers” produce the solutions of a given equation subsystem: merging the lists corresponds to merging two subsystems.

Lemma 1. *Let $\mathcal{R}[X_1]$ and $\mathcal{R}[X_2]$ be two reduced lists. From them, the reduced list $\mathcal{R}[X_1 \cup X_2]$ can be computed in time:*

$$\max(|\mathcal{R}[X_1 \cup X_2]|, |\mathcal{R}[X_1]|, |\mathcal{R}[X_2]|) . \quad (3)$$

Proof. Let Y be the set of linear equations of the system whose support is included in $X_1 \cup X_2$, but not in X_1 nor X_2 . Then by definition of reduced lists, we have: $|\mathcal{R}[X_1 \cup X_2]| = |\mathcal{R}[X_1]| \times |\mathcal{R}[X_2]| / (\sum_{L \in Y} \text{width}(L))$.

We separate each linear equation L of Y into its X_1 -part L_1 and its X_2 -part L_2 : the equation becomes $L(X) = L_1(X_1) \oplus L_2(X_2) = 0$. We compute $L_1(X_1)$ for all cell assignments in $\mathcal{R}[X_1]$, likewise we compute $L_2(X_2)$ for all cell assignments in $\mathcal{R}[X_2]$. We then sort both lists with respect to these values, and we look for collisions. The collision pairs are computed efficiently by iterating over both lists, and give the matching cell assignments of $\mathcal{R}[X_1 \cup X_2]$. \square

Remark 1. The merging operation is the same in the extended setting. In the AES-like setting, there can be implicit linear relations between cells. This corresponds to *matching through MixColumns*; we explain how we model this in [Section 4.3](#).

It can be shown by a trivial induction that, if [Assumption 1](#) holds for individual cells, then the sizes of all reduced lists are exactly powers of 2. Of course this is true only on average if we consider S-Boxes drawn at random. In practice, the S-Boxes are fixed, but the deviation from this average is small.

Definition. A *merging-based MITM attack* is a *merging strategy* represented by a binary tree \mathcal{T} , whose nodes are identified by sets of cells X , such that: • the leaves contain individual cells; • the root contains the set of all cells; • the set of cells of a given node is the union of the set of cells of its children. Then each node represents a reduced list. The attack consists in computing the reduced lists in any order consistent with the tree. By [Lemma 1](#), its time complexity is given by $\max_{X \in \mathcal{T}} |\mathcal{R}[X]|$.

The strategy of [\[16,22\]](#) is an exploration of the merging strategies, starting from individual cells and computing the complexity of reduced lists until enough cells are covered. Paths stop when the complexity exceeds the generic one. Thus, the dedicated solver that they use is also *bottom-up*, not in the definition of constraints like [\[4\]](#), but in the way it computes the complexity of possible attacks.

3.4 Global Edges

In all settings (PRESENT, AES, extended), an important extension of merging-based MITM attacks is the ability to guess globally the value of an edge. We use *global edges* in three cases.

Input-Output Constraints. To model input-output constraints, we create wrapping constraints and make these edges global. With this view, we remark that a MITM problem always has same or lower complexity with a given amount of wrapping constraints compared to the same amount of input-output constraints.

Reducing the Number of Solutions. In the PRESENT and AES-like setting, it can be seen that when the system admits more than 1 solution, we can set global edges of a combined width equal to the quantity of [Equation 2](#). As long as the width of global edges on a given cell does not exceed 1, there is on average a solution. (This is not true in the “extended” setting, where global edges can *a priori* create inconsistencies in the system and more care is required.)

Reducing the Memory. Global edges allow to reduce the size of intermediate lists in the merging strategy. We can easily prove that they do not allow to reduce the time complexity. If we consider a system with α global edges, that admits a solution with probability $2^{-\alpha}$, we can redo any merging strategy by removing these global edges: the size of lists increases by a factor 2^α at most. Since the time complexity is the maximum of list sizes (multiplied by the loop on global guesses), it stays the same in both cases.

Global edges correspond to the *initial structure* in previous works on MITM attacks. An interesting consequence of this remark is that the initial structure is actually not necessary to obtain the best time complexity: it suffices to share its components between the backward and forward paths.

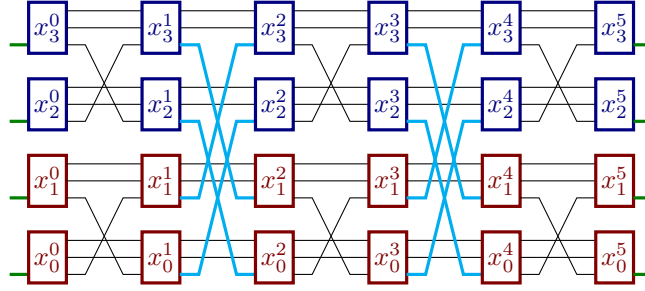


Fig. 4. 12-round MITM attack on GIMLI. The matching edges between the two final lists are highlighted in cyan.

3.5 Example: Gimli

Before we elaborate on our MILP modeling, we detail a simple example of a merging-based attack: the state-recovery on 12-round GIMLI-Cipher of [27].

GIMLI [8] is a cryptographic permutation with 384-bit state divided into 4 cells of 96 bits each. The full permutation has 24 rounds that apply an *SP-Box* to each cell individually, and then, every two round, perform a linear layer. The linear layer is either a *small swap* (32 bits are exchanged between cell 0 and 1, and between 2 and 3) or a *big swap* (32 bits are exchanged between cell 0 and 2, and between 1 and 3). In the cell-based representation, each cell has width 1, three input and three output branches of width 1/3 each, as can be seen in Figure 4. We do not need to consider the details of the SP-Box.

The attack of [27] targets GIMLI-Cipher, where GIMLI is used in a Duplex mode. The recovery of the internal state can be reduced to the following problem. Given the cell-based representation of Figure 4, where a single edge is fixed in the 4 input and output cells, the goal is to find the list of size $2^{4 \times 32} = 2^{128}$ (4/3 cells) of all possible values of the full state, in time less than 2^{256} (8/3 cells). The merging strategy is given in Figure 5, where the list sizes are computed in \log_2 and relatively to a cell. The time and memory complexities are 2^{192} (2 cells).

4 Simplification and MILP modeling

In Section 3 we have given a very generic definition of merging-based MITM attacks. We postulate that this definition contains all structural MITM attacks on permutations known to date. Unfortunately, this search space is too large for MILP solvers to be practical. Hence we consider a subset of these attacks, using only two lists, a *forward* list and a *backward* list. We motivate this definition in Section 4.1 and show how to obtain the list sizes from their cells. Then, we show in Section 4.2 how to obtain a MITM attack with a complexity determined by the list sizes. Finally in Section 4.3 we detail the MILP model itself.

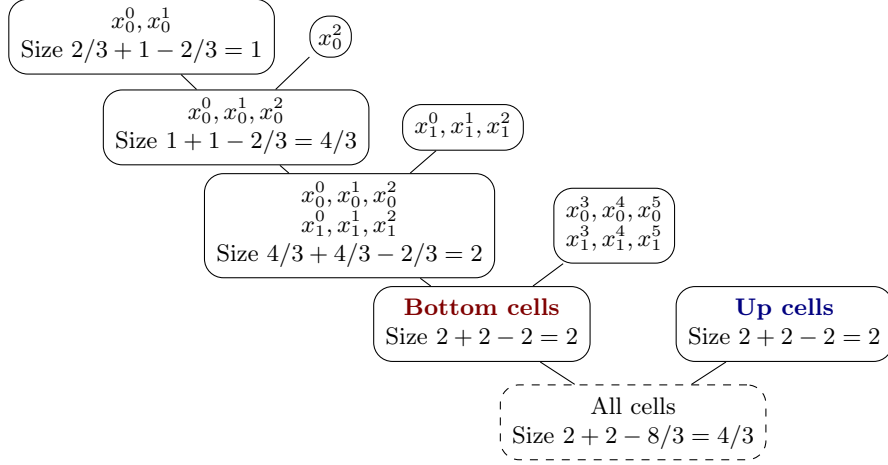


Fig. 5. Merging strategy for the 12-round attack against GIMLI-Cipher of [27]. Some lists are omitted by symmetry.

4.1 A Simpler Definition

In line with the “Meet-in-the-middle” terminology, we consider a merging strategy made of only three reduced lists: a *forward list* $\mathcal{R}[X_F]$, a *backward list* $\mathcal{R}[X_B]$ and a *merged list* $\mathcal{R}[X_F \cup X_B]$.

Forward and Backward Lists. Ultimately, the time complexity of the MITM attack is computed as a function of the list sizes, so we must define sets X_B and X_F in such a way that the list sizes $|\mathcal{R}[X_F]|$ and $|\mathcal{R}[X_B]|$ are simple functions of X_F and X_B respectively. In the generic binary trees, we used the fact that the size of leaves can be trivially computed (a list with a single cell of width w contains 2^w elements). Here we are simply making these leaves more complex, so that only two leaves are needed in the end.

Lemma 2. *Let X be a set of cells such that: (1) there is at least one round $0 \leq i \leq r - 1$ such that no cell x_j^i belongs to X ; (2) for every global linear constraint connecting cells $x_{j_1}^i$ and $x_{j_2}^{i+1}$, then only one of these two cells can be in X , and always either the one of the lower round number (backward case) or the upper round number (forward case). Let ℓ be the quantity:*

$$\ell = \sum_{x \in X} \left(w_x - \sum_{\substack{(x, x') \in E \\ x' \in X}} w_{(x, x')} \right) - \sum_{\substack{(x, x') \in E \\ x \in X \vee x' \in X \\ (x, x') \text{ is global}}} w_{(x, x')} , \quad (4)$$

then $\mathcal{R}[X]$ is of size exactly 2^ℓ . In the PRESENT-like / AES-like setting, it can be constructed in time 2^ℓ with negligible memory.

Here, *forward* and *backward* lists follow the same intuition as in standard MITM attacks, where there are two computational paths going into different directions. But this direction is only enforced because of global edges.

Proof. The size of a list is given by the sum of all widths of the cells, minus the linear constraints between them, minus the globally fixed edges. Since at least one round is cut, we can reorder the terms and associate all linear constraints (x, x') between round i and $i + 1$ to the cell x at round i : we obtain the formula for ℓ . Let us consider the backward list. We have:

$$\ell_B = \sum_{x \in X} \left(w_x - \sum_{\substack{(x, x') \in E \\ x' \in X}} w_{(x, x')} - \sum_{\substack{(x, x') \in E \\ (x, x') \text{ is global}}} w_{(x, x')} \right), \quad (5)$$

and we remark that each term is greater than zero: indeed, we cannot have $x' \in X$ and (x, x') global at the same time, by assumption, and in the PRESENT-like setting, we have $w_x \geq \sum_{(x, x') \in E} w_{(x, x')}$. This is not true in the extended setting (due to branching cells), but we can work around this in practice.

One constructs $\mathcal{R}[X]$ follows: separate X into X_{r-1}, \dots, X_0 , assuming that no cell is covered at round r . We start at round $r - 1$: we take values for all edges $(x, x') \in E$ with $x \in X_{r-1}$ that are not already global. Next, at round $r - 2$, we take values for all edges (x, x') with $x \in X_{r-2}$ that are neither connected to $x' \in X_{r-1}$, nor global. Each time, the number of bits to guess corresponds precisely to another term in ℓ . For the forward list, we rewrite ℓ as:

$$\ell_F = \sum_{x \in X} \left(w_x - \sum_{\substack{(x', x) \in E \\ x' \in X}} w_{(x', x)} - \sum_{\substack{(x', x) \in E \\ (x', x) \text{ is global}}} w_{(x, x')} \right), \quad (6)$$

and we change the direction of the procedure. This is a streaming procedure, which outputs the list elements without requiring any storage. In both cases, the list size corresponds exactly to the number of bits that we have to guess. \square

Simple Condition of Success. Initially, we required the merging strategy to compute the reduced list of all cells. However, we can stop as soon as *all cells can be deduced from the current list*. That is, given a valid sequence of values for the cells of $X_F \cup X_B$, we can deduce all the others without guessing new edges. Since we are studying a permutation, a sufficient condition (that we enforce) is that $X_F \cup X_B$ covers a complete round. (Intuitively, we dismiss the trivial merging steps consisting in adding the remaining cells one by one.)

Disjoint Paths. In the PRESENT- and AES-like settings (but not the “extended” setting), the sets X_F and X_B can be made disjoint at no loss. Since any intersection between X_F and X_B can either be removed from X_F , or from X_B , and in at least one case both list sizes decrease (the merged list remains unchanged).

4.2 From a Coloring to an Attack

Now we show that to any valid triple of sets $X_B, X_F, X_F \cup X_B$, there corresponds a MITM procedure whose time and memory complexities are determined solely by the size of the three lists involved. We use ℓ_B, ℓ_F, ℓ_M to denote the \log_2 of these list sizes, counted relatively to a cell. Our goal is to minimize this complexity. We assume for simplicity that the merging problem admits a single solution; it is easy to generalize this to multiple solutions in the classical setting.

Theorem 1. *Assume that X_B and X_F are defined as in [Lemma 2](#), and $X_F \cup X_B$ covers at least one round completely. Let g be the sum of all widths of global edges. Then there exists a classical and a quantum algorithm solving the MITM problem with the following complexities in \log_2 , relatively to a cell size. The classical algorithm has memory complexity $m_c = \min(\ell_F, \ell_B)$ and time complexity $t_c = g + \max(\ell_F, \ell_B, \ell_M)$. The quantum algorithm has memory $m_q = m_c = \min(\ell_F, \ell_B)$ and time complexity $t_q = \frac{g}{2} + \max(\min(\ell_F, \ell_B), \frac{1}{2} \max(\ell_F, \ell_B, \ell_M))$.*

Proof (sketch). In the classical setting, both leaf lists can be computed on the fly, we only need to store one of them (the smallest). The memory complexity is thus (in \log_2) $\min(\ell_F, \ell_B)$ and the time complexity $g + \max(\ell_F, \ell_B, \ell_M)$ (we must repeat the merging for every choice of global edges). One should note that by the definition of the leaf lists, there is no variance in their size. There can be a variance in the merged list size, which is usually dismissed in classical analyses.

Given a path for a two-list MITM attack, we can also write down a quantum algorithm to solve it. In short, this algorithm creates the smallest list (e.g., the forward one), then performs a Grover search in the merged list for a solution. We refer to the full version of the paper [\[45\]](#) for technical details. The algorithm requires quantum-accessible quantum memory (QRAQM). Assuming a single solution, the quantum time complexity can be bounded by:

$$2 \left(\frac{\pi}{4} 2^{g/2} + 1 \right) \left(2^{\ell_F} + \left(\frac{\pi}{4} \sqrt{2^{\ell_B}} + 1 \right) \left(\frac{\pi}{\sqrt{2}} \max \left(1, \sqrt{\frac{2^{\ell_M}}{2^{\ell_B}}} \right) + 6 \right) \right) \quad (7)$$

quantum evaluations of the attacked permutation, for a 1/2 chance of success. Asymptotically, this formula can be simplified into 2^{t_q} , where:

$$t_q = \frac{g}{2} + \max \left(\min(\ell_F, \ell_B), \frac{1}{2} \max(\ell_F, \ell_B, \ell_M) \right) , \quad (8)$$

which concludes the proof. \square

Criterion for a Quantum Attack. By comparing the quantum and classical time exponents, one can see that quantum attacks require an additional constraint compared to classical attacks: One can see that a classical MITM procedure constitutes an attack if $t_c < t$ where t is the generic time exponent to solve the MITM problem; in the quantum setting, this time is reduced by a square-root factor due to Grover search, so we need $t_q < t/2$. Unsurprisingly, any

quantum MITM attack turns into a classical attack: $t_q \leq t/2 \implies t_c \leq t$. In the other direction, if we have a valid classical path, and if the following additional constraint is satisfied: $\min(\ell_F, \ell_B) \leq \frac{1}{2} \max(\ell_F, \ell_B, \ell_M)$, then it also gives a valid quantum attack. This is true in particular when $\ell_M = 0$ and $\ell_F \leq \frac{1}{2} \ell_B$.

4.3 MILP Modeling

From the analysis above, we can see that we want to solve the problem:

Minimize the complexity formulas of [Theorem 1](#), under the constraints on X_F and X_B given by [Lemma 2](#), and the constraint that $X_F \cup X_B$ covers at least one round completely.

Our MILP model essentially uses boolean variables to represent X_F and X_B , continuous variables to represent global edges, and expresses the list sizes ℓ_F , ℓ_B , and ℓ_M depending on these variables. This model can be generated from the weighted graph (N, E) defined in [Section 3](#).

Present-like Setting: Variables. We start with the basic PRESENT-like constraints and explain afterwards the extensions. For each cell x , we introduce boolean *coloring* variables $\text{col}_F[x]$, $\text{col}_B[x]$ and $\text{col}_M[x]$ to represent the sets X_F , X_B and $X_M := X_F \cup X_B$. We have the constraint $\text{col}_M[x] = \max(\text{col}_F[x], \text{col}_B[x])$.

We constrain some round to be absent from X_F (resp. X_B), it can be chosen manually or not. For each edge (x, x') , we introduce a variable $\text{global}[x, x']$ which is 1 if the edge is globally guessed, 0 otherwise. It can be relaxed to a continuous variable. We constrain $X_F \cup X_B$ to cover at least one round entirely (chosen manually or not). Finally, we impose that for each edge (x, x') :

$$\begin{aligned} \text{col}_F[x] &\leq 1 - \text{global}[x, x'] & \text{col}_B[x'] &\leq 1 - \text{global}[x, x'] \\ \text{col}_B[x] &\geq \text{global}[x, x'] & \text{col}_F[x'] &\geq \text{global}[x, x'] \end{aligned}$$

Here the two constraints on the first line ensure that the conditions of [Lemma 2](#) are satisfied. The second line is not required, but it simplifies the formula for ℓ of [Lemma 2](#). Since each global constraints reduces the size of *both the forward and the backward lists*, we can introduce a term of *global reduction*:

$$g = \sum_{(x, x') \in E} \text{global}[x, x'] w_{x, x'} \quad , \quad (9)$$

which contains all of their contribution. At this point, we have defined a valid MITM strategy, and it only remains to compute the list sizes.

List Sizes. The list sizes are computed in \log_2 and relatively to the width of a cell (in practice cells may have different widths). For each list, there are two terms that intervene: the *contribution* of individual cells and the *global reduction*. For the forward list, following [Equation 6](#), we define the variables:

$$\text{contrib}_F[x] \geq w_x \text{col}_F(x) - \sum_{(x', x) \in E} w_{x', x} \text{col}_F(x') \quad , \quad (10)$$

and we have: $\ell_F = (\sum_{x \in N} \text{contrib}_F[x]) - g$. For the backward list, we define:

$$\text{contrib}_B[x] \geq w_x \text{col}_B(x) - \sum_{(x,x') \in E} w_{x,x'} \text{col}_B(x') \quad (11)$$

and we have similarly $\ell_B = (\sum_{x \in N} \text{contrib}_B[x]) - g$. For the merged list, we can go either forwards or backwards, for example:

$$\text{contrib}_M[x] \geq w_x \text{col}_M(x) - \sum_{(x,x') \in E} w_{x,x'} \text{col}_M(x'), \quad \ell_M = \sum_{x \in N} \text{contrib}_M[x] - g . \quad (12)$$

Since we have now expressed the list sizes, we implement the time and memory complexities using the formulas of [Theorem 1](#), e.g., classically:

$$\text{memory} = \min(\ell_F, \ell_B), \text{time} = g + \max(\ell_F, \ell_B, \ell_M) .$$

The primary optimization goal is the time and the secondary goal is the memory.

Extended Setting. In the extended setting, we must allow a *negative* contribution of the cells in each list. We have lower bounds: $\text{contrib}_F[x] \geq w_x - \sum_{(y,x) \in E} w_{y,x}$ and $\text{contrib}_B[x] \geq w_x - \sum_{(x,y) \in E} w_{y,x}$ which can be negative for branching cells. This is the only required change.

AES-like Setting. So far, our model considers the AES Super S-Box as a completely unknown function. We make two modifications to allow two techniques.

First, *matching through MC*. When we know $u \geq 4$ bytes in the input and output of an AES Super S-Box, we can reduce the merged list size by $u - 4$. Indeed, these edges are individual S-Boxes, and we can write linear equations between them using MixColumns. In order to model this, we modify the definition of $\text{col}_M[x]$. We authorize a cell of the merged list to be covered even if it does not belong to $X_F \cup X_B$, as soon as enough input and output edges are covered. This should not, however, happen at two successive rounds.

Second, *optimizing the memory through MC*. This is important for reaching better memory complexities on AES-like designs, but also, better quantum times. Assume that there exists a cell that belongs to the merged list but not the forward and backward ones. Assume that there are f_i input edges from the forward list, f_o output edges from the forward list, and respectively b_i and b_o such edges for the backward list. Recall that each edge here corresponds to an individual S-Box. Then we can add some shared constraints on these cells and make these constraints global. Indeed, if we know that: $\ell_1(x_0, x_1, x_2, x_3, y_0) = 0$ and $\ell_2(x_0, x_1, x_2, x_3, y_1) = 0$, we can create a global constraint $\ell'_1(x_0, x_1, x_2) = t$ and $\ell'_2(y_0, y_1) = t$. Going through MC, we can add up to $f_i + f_o + b_i + b_o - w$ such linear constraints, where w is the cell width in number of edges (4 in the case of AES). Furthermore, we need to have less such new constraints than f_o and b_i respectively: this ensures the existence of a streaming procedure for the lists and the validity of an adapted version of [Lemma 2](#).

Practical Improvements. Our code is more optimized than the presentation given in this section. In particular, we removed the `global[x, x']` variables attached to edges and replaced them by “global reduction” variables attached to each cell. These variables unify the PRESENT-like and AES-like settings, since they account both for the global edges and the reduction through MC.

Reducing the Search Space There are several ways to reduce the search space without affecting the optimality. First, we can prune the graph by removing cells that do not have both input and output edges (for example in the MITM attack on PRESENT of Section 5, many cells from the first and last rounds can be removed). Second, when two cells in the graph have the same forward and backward connections, their colorings can always be exchanged without changing the list sizes. This reduces massively the search space size in the case of highly symmetric AES-like designs, for example Grøstl-256 (see Section 6.3).

5 Application to PRESENT-like Permutations

Gimli. With our tool, we can prove the optimality of the 12-round state-recovery attack recalled in Section 3.5. Here our 2-list MILP model is not enough, since the two lists merged at level 1 in the tree span all the rounds. So, contrary to most of our examples, we used an extension to 4 lists.

Present and Spongent. The current best distinguishers on known-key PRESENT [10] and reduced-round SPONGENT- π [47] combine a MITM layer and a truncated differential layer. By improving the MITM layer, we improve indirectly the number of rounds that can be targeted.

In a nutshell, the goal is to construct the list of 2^{56} input states that satisfy a 4-bit input constraint and a 4-bit output constraint, in time less than 2^{60} . In [10] the constraint is put at position 13; for SPONGENT- π we tried the position 0. We conjecture that due to the high amount of symmetries in the design, the number of attacked rounds should remain the same independently of this position.

The MITM layer for [10] reaches 7 rounds, in time 2^{56} and memory 2^{32} . The time is optimal, but we improve the memory to 2^{12} . Next, we find an attack with one more round. The time complexity then rises to 2^{58} (14.5 cells) and the memory complexity to 2^{43} (10.75 cells). In order to make the optimization converge, we used the following simplification: we merged pairwise the cells of the middle rounds. These pairs of cells thus have the same coloration; this simplification reduces greatly the number of variables, while still allowing interesting results.

Spongent. This strategy was extended in [47] to the SPONGENT- π permutations, which are used in the hash function SPONGENT [11] and the permutation-based AEAD Elephant [9] (in the “Dumbo” version). Following [47, Table 1], we denote the number of rounds of both phases (truncated differential and MITM) by r_0 and r_1 and report them in Table 2, where our new results appear in **bold** in the last column. The table contains all state sizes specified in [11,12,9]. Here

Table 2. Versions of SPONGENT, results from [47] and our improvements.

	State size (bits)	Rounds Attacked / full	Cells	r_0	r_1	New r_0
PRESENT	64	31 / 31	16	7	24	8 (+ 1)
SPONGENT-88/80/8	88	30 / 45	22	7	23	8 (+ 1)
SPONGENT-128/128/8	136	43 / 70	34	7	36	8 (+ 1)
SPONGENT- π [160]	160	80	40			9
SPONGENT-160/160/16	176	53 / 90	44	7	46	9 (+ 2)
SPONGENT-160/160/80	240	69 / 120	60	7	62	10 (+ 3)
SPONGENT-88/176/88	264	77 / 135	66	9	68	10 (+ 1)
SPONGENT-256/256/16	272	68 / 140	68	9	69	10 (+ 1)
SPONGENT-224/224/112	336	95 / 170	84	9	86	10 (+ 1)
SPONGENT-128/256/128	384	109 / 195	96	11	98	11
SPONGENT-160/320/160	480	132 / 240	120	9	123	12 (+ 3)
SPONGENT-224/448/224	672	181 / 340	168	9	172	12 (+ 3)
SPONGENT-256/512/256	768	192 / 385	192	11	194	12 (+ 1)

the notation SPONGENT- $n/c/r$ refers to [12], where n is the output hash size, c the capacity and r the rate, while SPONGENT- π refers to the permutation itself. The 160-bit version used in Elephant [9] was not studied previously, because SPONGENT- π [160] does not appear among the different parameterizations of the SPONGENT hash functions.

As in [10], the MITM layer finds all the input-output pairs such that: 4 bits of an S-Box are fixed in input, and in output, to arbitrary values. The generic complexity would be 2^{b-4} evaluations of the permutation. The lowest complexity possible is 2^{b-8} since this is the number of solutions. Since the state size becomes quite large, we do not use our tool as an optimization, but rather as a solver: we set the minimal complexity 2^{b-8} as optimization goal and kill the process if it runs for too long (say, 500 seconds). By our experiments, we expect solutions to be found quite quickly, if they exist.

6 Application to AES-based Permutations

As remarked above, our model does not include degrees of freedom of the key-schedule, and some of the previous preimage attacks on AES-like hashing cannot be recovered. However, all known results on AES-based permutations [3,4,5,25], *except* the non-linear computation of neutral words proposed in [5] (see the example of Grøstl below), can be recovered by our simplified modeling. We only present new attacks obtained by our tool in this section. In the classical setting, we improve the attack on Haraka-512 v2 of [4]. In the quantum setting, we give attacks on reduced-round AES, Haraka and Grøstl.

Note that an AES-like state is an $n \times m$ matrix of bytes, which we represent as m cells with n input and output edges. The SR operation moves individual

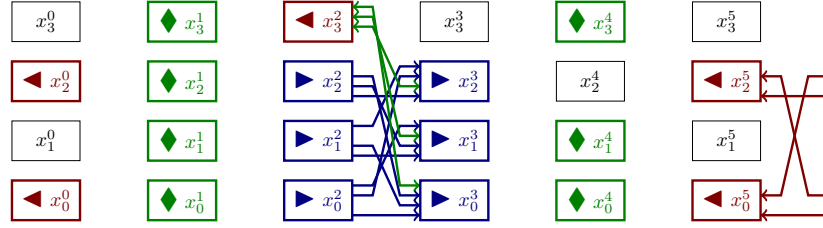


Fig. 6. AES 7-round quantum attack. \blacktriangleleft : backward, \blacktriangleright : forward, \blacklozenge : matching through MC (new cells in the merged list), \leftrightarrow : global edges.

bytes between the cells. When the last MC operation is omitted, and round $r - 1$ is connected to round 0, then round 0 can actually be bypassed. Indeed, the columns at the beginning of round 1 (before SB), which correspond to the cells at round 1 in our representation, can immediately be linked to the columns at round $r - 1$ (which correspond to the cells at round $r - 1$). Though MC has been removed, we usually keep the last SR operation; this creates a special round in which bytes are exchanged between pairs of cells only.

6.1 Quantum Attack on 7-round AES

On this example, like the following ones, our attack is a pseudo-preimage attack that, given a target t , finds x such that $x \oplus AES(x) = t$. None of the attacks known classically can be adapted in the quantum setting (they don't satisfy the condition given in Section 4.2), so we use our tool to find a new optimization. The path is displayed in Figure 6.

Details of the Attack. We count the complexities in cells. The attack has 2.75 global guesses, with 0.75 global edges and 2 additional reductions through MC at round 1. For each of these 2.75 choices, we compute the three lists.

First, the **backward** \blacktriangleleft list is of size 0.25. We start by x_3^2 which contributes only to 0.25. We move to x_0^0 and x_2^0 which are entirely determined by the reduction through MC of round 1. We deduce x_5^0, x_2^5 . Second, the **forward** \blacktriangleright list is of size 1. We start by x_0^2, x_1^2, x_2^2 , which have only $3 - 2 = 1$ degree of freedom by the reduction through MC of round 1. We deduce x_0^3, x_1^3, x_2^3 . Third, the merged list is of size ≤ 1 . We match through MC at round 4, each cell gives 0.25 degree of matching, so one would be enough.

This corresponds to an attack of classical time 2^{120} and memory 2^8 , so equivalent to the attack of [44]. However, using Equation 8, we obtain a quantum time 2^{60} , and with the precise formula of Equation 7, we have a time of $2^{63.34}$ quantum evaluations of the primitive (Grover search would stand at $2^{64.65}$).

6.2 New Attacks on Haraka v2

Haraka v2 [40] is a short-input AES-like hash function intended for use within post-quantum signature schemes based on hash functions, such as SPHINCS+ [2].

There are two variants: (1) **Haraka-256 v2** hashes 256 bits to 256 using a 256-bit permutation in feed-forward mode: $x \mapsto \pi_{256}(x) \oplus x$; (2) **Haraka-512 v2** hashes 512 bits to 256 using a 512-bit permutation with a truncation: $x \mapsto \text{trunc}(\pi_{512}(x) \oplus x)$. The internal state of **Haraka-256 v2** (resp. -512) is the concatenation of 2 (resp. 4) AES states. The columns of these states are numbered from 0 to 7 (resp. 0 to 15). Each **Haraka** round (total 5) applies two AES rounds (AC, SB, SR, MC) individually on the states, followed by a MIX operation which permutes the columns:

$$\begin{aligned} \text{MIX}_{512} &: 0, \dots, 15 \mapsto (3, 11, 7, 15), (8, 0, 12, 4), (9, 1, 13, 5), (2, 10, 6, 14) \\ \text{MIX}_{256} &: 0, \dots, 7 \mapsto (0, 4, 1, 5), (2, 6, 3, 7) \end{aligned}$$

The truncation `trunc` extracts the columns (2, 3, 6, 7, 8, 9, 12, 13).

Integration in SPHINCS, SPHINCS+ and Attacks. In [39], Kölbl proposed to integrate **Haraka** into SPHINCS. Here both **Haraka-256 v2** and **Haraka-512 v2** need 256 bits of classical preimage security and 128 bits of quantum preimage security (see [39], Section 3). In [4], the authors found a classical 4.5-round preimage attack on **Haraka-256 v2** and a 5.5-round attack (extended by 0.5 round) on **Haraka-512 v2**. None of the attacks of [4] apply directly to the post-quantum signature scheme SPHINCS+ [2], an “alternate” finalist of the NIST post-quantum standardization process. Here **Haraka-512** is used in a Sponge with 256 bits of rate and 256 bits of capacity. The targeted security level is 128 bits due to a generic second-preimage attack. We obtain a classical MITM attack on 4.5 rounds of complexity 2^{192} , and a quantum preimage attack on 3.5 rounds of complexity 2^{64} . The details are provided in the full version of the paper [45].

New Quantum Attack on Haraka-256 v2. The attack path of [4] does not meet our criteria for quantum attacks, since both the forward and backward lists have size 1 cell, and the total time complexity is 7 cells. However, a reoptimization allows to reach an attack with 5 global guesses, a forward list of size 2, a backward list of size 1 and a merged list of size 2 (details in the full version of the paper [45]). By Eq. 7, this gives a quantum time $2^{115.55}$ against a generic 2^{128} .

Improved Attack on Haraka-512 v2. The 5.5 round attack of [4] has time complexity 2^{240} (7.5 cells) and memory complexity 2^{128} (4 cells). In order to make our optimization converge faster, we constrain the pattern in the first and last rounds to contain full active AES states, like in [4]). We obtain the path of **Figure 7**, which reduces the memory down to 0.5 cell (2^{16}). The main difference with the framework of [4] is that the matching occurs in several rounds separately.

We first guess **28 bytes** \bullet : $x_3^a[0, 1, 5, 6, 10, 11, 12, 15]$, $x_3^d[10, 11, 12, 15]$, $x_4^a[4-11]$, $x_4^b[0, 1, 2, 3, 12, 13, 14, 15]$, and we precompute two linear relations between the first and second columns of z_2^d and w_2^d , and one linear relation for each column between z_6 and w_6 . The total is 46 bytes, i.e., 11.5 cells, of global guesses (including 8 for free). Then for the **forward ► list** (size 0.5 cells), we start from w_2^d . We have 4 bytes and two precomputed linear relations, thus 2 bytes of freedom. We continue to compute until z_6 . In each column, we have one byte

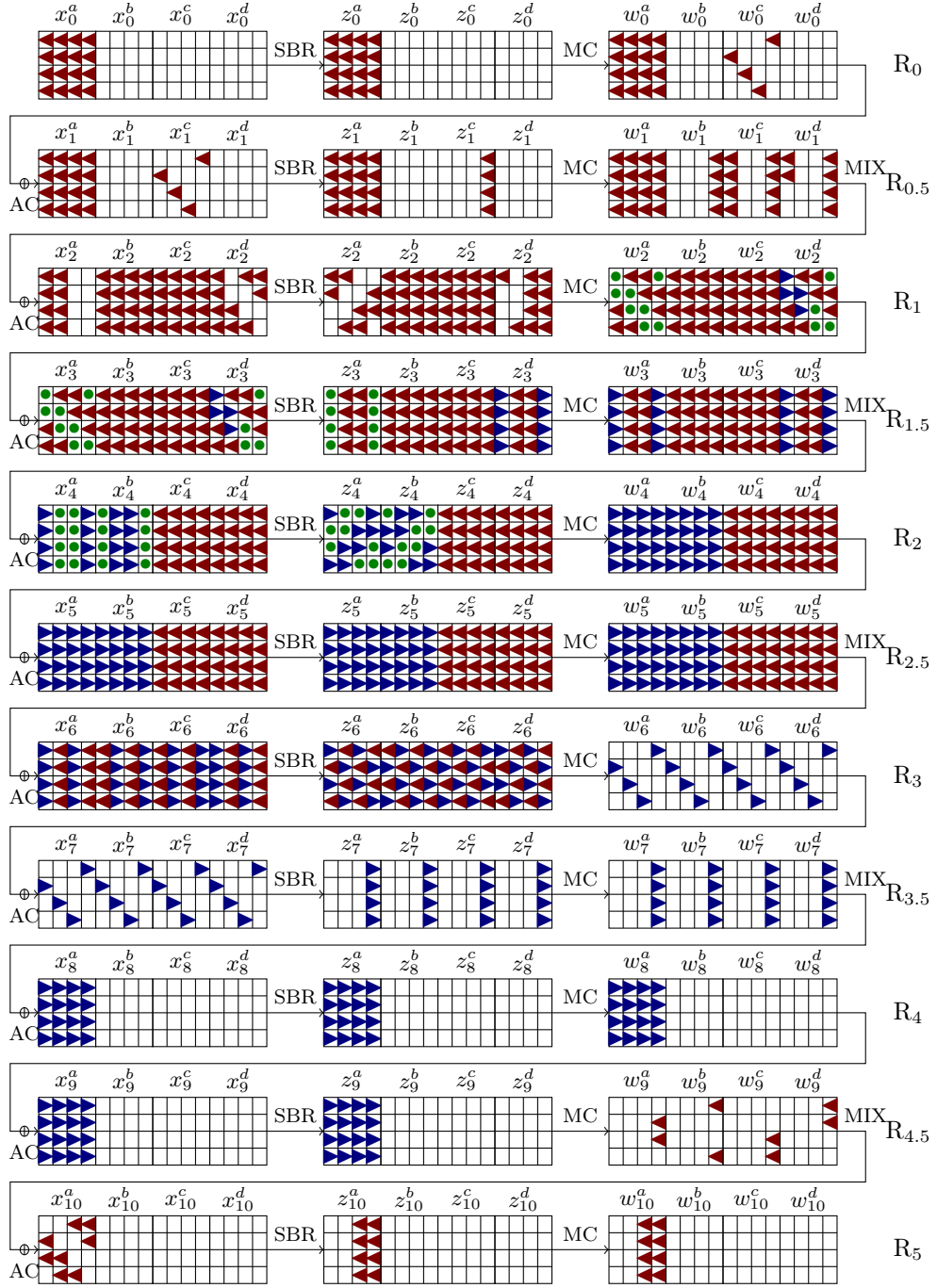


Fig. 7. Path of our improved attack on Haraka-512 v2. ◀: backward, ▶: forward, ●: guessed

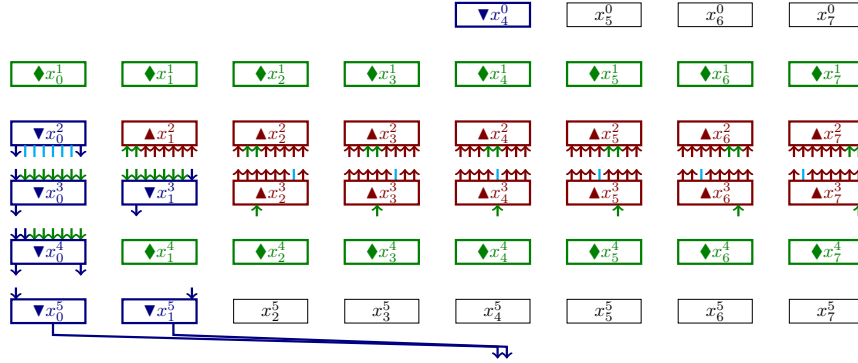


Fig. 8. Path of the quantum attack on Grøstl-256 OT. **▲: backward**, **▼: forward**, **◆: matching** through MC (new cells in the merged list), **↔: global edges**.

of precomputed linear relation, thus we can deduce all the blue bytes in w_6 immediately. We continue until x_9 . Next, for the **backward ◀ list** (size 4 cells), we start from z_6 . There are 32 red bytes and 16 precomputed linear relations, thus 16 bytes (4 cells) of freedom. From there we can compute backwards until w_2 . We deduce $z_2^d[0, 7]$ using the two precomputed relations, and the rest by direct computation. We compute until w_{10}^a . Finally, a matching of more than 2 bytes occurs between rounds 9 and 10. With these lists, the classical time stays at 2^{240} . By Equation 7, the quantum time is $2^{123.34}$, against 2^{128} generically.

This attack of large complexity also yields a practical *partial preimage attack* that finds x such that $MC^{-1}(x \oplus \pi(x))$ has 32 bits to zero, in about 2^{16} evaluations of Haraka-512 v2. We just have to run a single merging step, fixing the global variables. For each choice of forward and backward values in the merged list, we recompute the initial state x . This x is such that the two cells $MC^{-1}(x \oplus \pi(x))^a[10, 13]$ are zero. Since the merged list is of size 2^{16} , by enumerating it in times 2^{16} , we will find an element with 16 more zero bits.

6.3 Quantum Attack on Grøstl OT

The output transformation (OT) of Grøstl-256 [28] is an AES-like permutation P operating on an 8×8 matrix of bytes (thus 512 bits in total). The goal is to find a state x such that $\text{trunc}_{256}(P(x) \oplus x) = t$ for some target value t , say zero. The generic complexity is 2^{256} .

With our tool, we can recover the 6-round attack of [25, Appendix D]. We can also recover the improved time complexity of [5] (2^{224} , 3.5 cells), but not its memory complexity, because their procedure for the backward list is more complex than a streaming procedure. We obtain only a memory 2^{224} .

New Quantum Attack. We do not know if the approach of [5] could lead to a quantum attack, as they require a memory of size 2^{128} : in the quantum setting, one cannot afford a precomputation of time 2^{128} since it already becomes larger

than the limit given by Grover search. By optimizing for the quantum time complexity, our tool finds the path of [Figure 8](#). There are 4.25 global guesses (including 4 free guesses), with 2.5 cells of global linear constraints and 0.25 reduction through MixColumns in each of the 7 green cells at round 4. First, the **forward ▼ list** (1.75 cells): we start from x_0^2 . We deduce immediately the blue cells at round 3 and 4. Then using the 1.75 cells of precomputed equations at round 4, we deduce completely x_0^5 and x_1^5 . There remains 6 bytes (0.75 cells) to guess to obtain x_4^0 . Second, the **backward ▲ list** (3.5 cells): we start from round 3. With the 1.75 precomputed equations and 0.75 guessed values, there remain 3.5 cells of freedom. We deduce completely the cells at round 2. Finally, matching: there is 0.75 cell of matching between round 2 and 3 and 1 cell of matching through MC between round 0 and round 2, so 1.75 in total, which gives a merged list of size 3.5 cells. By [Equation 7](#), the quantum time is $2^{123.56}$, against 2^{128} generically.

For 8 rounds of Grøstl-512, there are no symmetries anymore, and the model becomes quite large. We simplify it by merging the cells in groups of 4. Then, we use the results as “hints” for the detailed version. We reobtain the time complexity of [\[5\]](#) with a corresponding memory complexity of 2^{304} (instead of 2^{224}), and we find a quantum attack detailed in the full version of the paper [\[45\]](#).

7 Applications to Feistel Networks

7.1 GFNs and Simpira

The extended setting that we defined in [Section 3](#) allows to model a large class of permutations, and in particular, GFNs and SPARKLE.

Simpira v2 (simply Simpira in what follows) is a family of permutations proposed in [\[31\]](#). For each $b \geq 2$, Simpira- b is a b -branch GFN where each branch is a 128-bit AES state. (Though in contrast to a GFN, the branches are not swapped and the round functions are simply applied in place). Simpira-2 is a standard FN, Simpira-3 a 3-branch type-I GFN in the classification of [\[48\]](#), Simpira-4 a type-II 4-branch GFN, Simpira-6 and Simpira-8 have structures taken from [\[46\]](#). Each round function performs 2 complete rounds of AES with a certain round constant; we use II_i to denote them (where i indicates the current round constant). Examples for Simpira-2, -3 and -4 are depicted in the full version of the paper [\[45\]](#).

For $b \geq 2$, we can use any permutation in the family to define a small-range hash function G_b by feed-forwarding:

$$G_b : \begin{cases} \{0, 1\}^{b \times 128} & \rightarrow \{0, 1\}^{256} \\ x & \mapsto \text{trunc}(\text{Simpira-}b(x) + x) \end{cases} \quad (13)$$

where trunc is the truncation to the first 256 bits. The proposal SPHINCS-Simpira [\[32\]](#) uses G_2 and G_4 in SPHINCS+.

The authors of Simpira claim only 128-bit preimage security for the functions G_b , although the generic classical preimage search would stand at a time 2^{256} .

Table 3. Distinguishers on *Simpira*: number of rounds attacked (rounds / total rounds) by our automatic MITM tool, and by a dedicated GAD approach.

b	2	3	4	6	8
MITM (automatic)	Inapplicable	8 / 21	7 / 15	9 / 15	9 / 18
GAD	5 / 15	11 / 21	9 / 15	9 / 15	9 / 18

SPHINCS-*Simpira* [32] also claims 128-bit *quantum* preimage security. The quantum security of *Simpira* was studied in [42], but only regarding collision attacks. Among the known results on unkeyed GFNs, e.g. a 5-round distinguisher on a 2-branch FN [18] and a 8-round distinguisher on the 4-branch, type-II GFN [17], we did not find immediate preimage attacks on the G_b .

Results of the Extended Model. By making no structural assumption on the round functions, our model represents any GFN as a directed graph of 2-XOR cells (corresponding to round functions) and 2-branching cells. We may add *dummy cells* (1-branching cells) to separate clearly the rounds. The round functions do not need to be permutations; the attacks have a complexity at least the size of one branch, which is the cost of inverting a round function by brute force.

In order to maximize the number of rounds attacked, we consider a full wrapping constraint. We remove the memory optimization: we look for attacks of time and memory complexity $2^{(b-1)w}$ against generic 2^{bw} , where w is the branch width. Then, we run our tool with a 4-list MILP model. The results are reported in Table 3.

7.2 Guess-and-determine Attacks on GFNs

We remarked that, with the *Simpira*- b structures for $b \leq 8$, we could attack the same number of rounds, and more, using much simpler Guess-and-determine (GAD) attacks. These results are also given in Table 3. The increased number of rounds is due to the linearity of the XOR, which is not captured by our cell-based modeling (see *Simpira*-2 below).

These attacks are *partial preimage* attacks on the hash functions G_b . We find x such that $G_b(x) = 0_{128}|*$. From there, we have a full preimage of G_b in classical time 2^{128} and quantum time 2^{64} , still valid if we replace the Π_i by random functions F_i (we can invert the F_i by brute force).

Example: Simpira-2. We explain our strategy with a 5-round attack on *Simpira*-2 (see Fig. 16 in the full version of the paper [45]). We index the branches as follows: first, the initial state is named S_0, \dots, S_{b-1} . Then, each time a new operation $S_i \leftarrow S_i \oplus F(S_j)$ is applied, the resulting state is named S_k , with the current index k (which is then incremented). So we want to solve the following equation system:

$$\begin{aligned} S_1 \oplus S_2 &= \Pi_1(S_0), & S_2 \oplus S_4 &= \Pi_3(S_3) & S_0 \oplus S_3 &= \Pi_2(S_2) \\ S_0 \oplus S_3 &= \Pi_4(S_4), & S_4 \oplus S_6 &= \Pi_5(S_5), & S_0 &= S_5 \text{ (wrapping)} \end{aligned} .$$

Table 4. Distinguishers on SPARKLE. * The attacks from Table 4.9 in [6], can be extended by one step when attacking the permutation instead of the AEAD mode.

Target	Type	Steps	Time	Generic time	Memory	Source
SPARKLE-256	Classical	5 / 10	2^{96}	2^{128}	2^{96}	[6] *
SPARKLE-384	Classical	5 / 11	2^{128}	2^{192}	2^{128}	[6] *
SPARKLE-512	Classical	5 / 12	2^{192}	2^{256}	2^{160}	[6] *
SPARKLE-256	Practical	4 / 10	negl.	2^{64}	negl.	This paper
SPARKLE-384	Practical	4 / 11	negl.	2^{64}	negl.	This paper
SPARKLE-512	Practical	5 / 12	$< 2^{32}$	2^{64}	negl.	This paper

As we can see, there are 6 equations and 7 variables, since we have put a wrapping constraint on one branch. We can simplify this system by removing all variables that intervene in a single equation, i.e., S_6 and S_1 . We obtain:

$$S_0 \oplus S_3 = \Pi_2(S_2), \quad S_2 \oplus S_4 = \Pi_3(S_3), \quad S_0 \oplus S_3 = \Pi_4(S_4) .$$

From this we obtain the new equation $\Pi_4(S_4) = \Pi_2(S_2)$, which is not captured by our cell-based modeling. Guessing S_4 (our only degree of freedom) we can deduce S_2 , and all the other variables follow. After trying for $b = 2, 3, 4, 6, 8$, we found that this expansion of the equation system was only useful for *Simpira-2* and *Simpira-4*. The appropriate internal guesses are found automatically using another automated tool, which would work for any GFN construction.

7.3 Application to Sparkle

SPARKLE is a family of permutations upon which the NIST LWC candidate SCHWAEMM / ESCH (respectively for AEAD and hashing) [7] is based. We refer to the submission document [6] for a complete specification of SPARKLE, since we abstract out most of its components.

There exists three variants SPARKLE-256, -384 and -512, with respectively $b = 4, 6$ and 8 branches of 64 bits. One step of SPARKLE has the following operations: (1) an ARX-box (using round constants to disrupt symmetries) is applied to all branches. (2) a linear function of the $b/2$ left branches is computed (noted ℓ' in [6], and L here). (3) each left branch $i \leq b/2$ is XORed to branch $i + b/2$; the output of L is also XORed to each branch $i + b/2$. (4) the $b/2$ right branches are swapped following a standard GFN pattern, and then, the groups of left and right branches are swapped.

SPARKLE is not a GFN since the “round function” is actually linear, and the non-linear functions (the ARX boxes) are computed alongside the branches. But this makes no difference for our extended representation. We obtain results similar to *Simpira*: the MILP solver finds 4-step MITM distinguishers on the 3 variants of the permutation, and these can be simplified and improved with a GAD strategy. The details are given in the full version of the paper [45].

Our results are summarized in [Table 4](#). We found a GAD distinguisher of complexity 1 for 4-step SPARKLE-256 and -384, and a practical 5-step distinguisher for SPARKLE-512, which combines the GAD strategy with SAT solving. It highlights another limitation of our automatic approach: the ARX boxes are viewed as random permutations, although solving some ARX equations can be done practically.

As a comparison, the *birthday-differential* GAD attacks given in the NIST submission document [6], which break 4 steps in the authenticated encryption mode SCHWAEMM, can also be turned into 5-step distinguishers for the permutation. But they have large complexities, and our distinguishers are the first practical ones.

Acknowledgments. We want to thank Patrick Derbez and Léo Perrin for helpful discussions, and Xavier Bonnetain for contributing to the code used for generating some figures of this document. A.S. is supported by ERC-ADG-ALGSTRONGCRYPTO (project 740972).

References

1. Aoki, K., Sasaki, Y.: Preimage attacks on one-block MD4, 63-step MD5 and more. In: Selected Areas in Cryptography. LNCS, vol. 5381, pp. 103–119. Springer (2008)
2. Aumasson, J.P., Bernstein, D.J., Beullens, W., Dobraunig, C., Eichlseder, M., Fluhrer, S., Gazdag, S.L., Hülsing, A., Kampanakis, P., Kölbl, S., Lange, T., Lauridsen, M.M., Mendel, F., Niederhagen, R., Rechberger, C., Rijneveld, J., Schwabe, P., Westerbaan, B.: SPHINCS+: Submission to the NIST post-quantum project
3. Bao, Z., Ding, L., Guo, J., Wang, H., Zhang, W.: Improved meet-in-the-middle preimage attacks against AES hashing modes. IACR Trans. Symmetric Cryptol. **2019**(4), 318–347 (2019)
4. Bao, Z., Dong, X., Guo, J., Li, Z., Shi, D., Sun, S., Wang, X.: Automatic search of meet-in-the-middle preimage attacks on AES-like hashing. In: EUROCRYPT (1). LNCS, vol. 12696, pp. 771–804. Springer (2021)
5. Bao, Z., Guo, J., Shi, D., Tu, Y.: MITM meets guess-and-determine: Further improved preimage attacks against aes-like hashing. IACR Cryptol. ePrint Arch. **2021**, 575 (2021)
6. Beierle, C., Biryukov, A., dos Santos, L.C., Großschädl, J., Perrin, L., Udovenko, A., Velichkov, V., Wang, Q.: SCHWAEMM and ESCH: lightweight authenticated encryption and hashing using the SPARKLE permutation family. Submission to the NIST lightweight standardization process (second round). (2019)
7. Beierle, C., Biryukov, A., dos Santos, L.C., Großschädl, J., Perrin, L., Udovenko, A., Velichkov, V., Wang, Q.: Lightweight AEAD and hashing using the SPARKLE permutation family. IACR Trans. Symmetric Cryptol. **2020**(S1), 208–261 (2020)
8. Bernstein, D.J., Kölbl, S., Lucks, S., Massolino, P.M.C., Mendel, F., Nawaz, K., Schneider, T., Schwabe, P., Standaert, F., Todo, Y., Viguier, B.: Gimli: A cross-platform permutation. In: CHES. LNCS, vol. 10529, pp. 299–320. Springer (2017)
9. Beyne, T., Chen, Y.L., Dobraunig, C., Mennink, B.: Dumbo, jumbo, and delirium: Parallel authenticated encryption for the lightweight circus. IACR Trans. Symmetric Cryptol. **2020**(S1), 5–30 (2020)

10. Blondeau, C., Peyrin, T., Wang, L.: Known-key distinguisher on full PRESENT. In: CRYPTO (1). LNCS, vol. 9215, pp. 455–474. Springer (2015)
11. Bogdanov, A., Knezevic, M., Leander, G., Toz, D., Varici, K., Verbauwhede, I.: Spongent: A lightweight hash function. In: CHES. LNCS, vol. 6917, pp. 312–325. Springer (2011)
12. Bogdanov, A., Knezevic, M., Leander, G., Toz, D., Varici, K., Verbauwhede, I.: SPONGENT: the design space of lightweight cryptographic hashing. *IEEE Trans. Computers* **62**(10), 2041–2053 (2013)
13. Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y., Vikkelsoe, C.: PRESENT: an ultra-lightweight block cipher. In: CHES. LNCS, vol. 4727, pp. 450–466. Springer (2007)
14. Bogdanov, A., Rechberger, C.: A 3-subset meet-in-the-middle attack: Cryptanalysis of the lightweight block cipher KTANTAN. In: *Selected Areas in Cryptography*. LNCS, vol. 6544, pp. 229–240. Springer (2010)
15. Bogdanov, A., Shibutani, K.: Double SP-functions: Enhanced Generalized Feistel Networks - extended abstract. In: ACISP. LNCS, vol. 6812, pp. 106–119. Springer (2011)
16. Bouillaguet, C., Derbez, P., Fouque, P.: Automatic search of attacks on round-reduced AES and applications. In: CRYPTO. LNCS, vol. 6841, pp. 169–187. Springer (2011)
17. Chang, D., Kumar, A., Sanadhya, S.K.: Security analysis of GFN: 8-round distinguisher for 4-branch type-2 GFN. In: INDOCRYPT. LNCS, vol. 8250, pp. 136–148. Springer (2013)
18. Coron, J., Patarin, J., Seurin, Y.: The random oracle model and the ideal cipher model are equivalent. In: CRYPTO. LNCS, vol. 5157, pp. 1–20. Springer (2008)
19. Daemen, J., Rijmen, V.: AES Proposal: Rijndael. Submission to the NIST AES competition (1999)
20. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. *Information Security and Cryptography*, Springer (2002)
21. Damgård, I.: A design principle for hash functions. In: CRYPTO. LNCS, vol. 435, pp. 416–427. Springer (1989)
22. Derbez, P., Fouque, P.: Automatic search of meet-in-the-middle and impossible differential attacks. In: CRYPTO (2). LNCS, vol. 9815, pp. 157–184. Springer (2016)
23. Diffie, W., Hellman, M.E.: Special feature exhaustive cryptanalysis of the NBS data encryption standard. *Computer* **10**(6), 74–84 (1977)
24. Dobraunig, C., Eichlseder, M., Mendel, F., Schl affer, M.: Ascon v1.2. Submission to NIST-LWC (2nd Round) (2019)
25. Dong, X., Hua, J., Sun, S., Li, Z., Wang, X., Hu, L.: Meet-in-the-middle attacks revisited: Key-recovery, collision, and preimage attacks. In: CRYPTO (3). LNCS, vol. 12827, pp. 278–308. Springer (2021)
26. Dunkelman, O., Sekar, G., Preneel, B.: Improved meet-in-the-middle attacks on reduced-round DES. In: INDOCRYPT. LNCS, vol. 4859, pp. 86–100. Springer (2007)
27. Fl orez-Guti errez, A., Leurent, G., Naya-Plasencia, M., Perrin, L., Schrottenloher, A., Sibleyras, F.: Internal symmetries and linear properties: Full-permutation distinguishers and improved collisions on Gimli. *J. Cryptol.* **34**(4), 45 (2021)
28. Gauravaram, P., Knudsen, L.R., Matusiewicz, K., Mendel, F., Rechberger, C., Schl affer, M., Thomsen, S.S.: Gr ostl-a SHA-3 candidate. Submission to the SHA-3 competition (2011)

29. Gleixner, A., Bastubbe, M., Eifler, L., Gally, T., Gamrath, G., Gottwald, R.L., Hendel, G., Hojny, C., Koch, T., Lübbecke, M.E., Maher, S.J., Miltenberger, M., Müller, B., Pfetsch, M.E., Puchert, C., Rehfeldt, D., Schlösser, F., Schubert, C., Serrano, F., Shinano, Y., Viernickel, J.M., Walter, M., Wegscheider, F., Witt, J.T., Witzig, J.: The SCIP Optimization Suite 6.0. Technical report, Optimization Online (2018)
30. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: STOC. pp. 212–219. ACM (1996)
31. Gueron, S., Mouha, N.: Simpira v2: A family of efficient permutations using the AES round function. In: ASIACRYPT (1). LNCS, vol. 10031, pp. 95–125 (2016)
32. Gueron, S., Mouha, N.: SPHINCS-Simpira: fast stateless hash-based signatures with post-quantum security. IACR Cryptol. ePrint Arch. p. 645 (2017)
33. Guo, J., Ling, S., Rechberger, C., Wang, H.: Advanced meet-in-the-middle preimage attacks: First results on full tiger, and improved results on MD4 and SHA-2. In: ASIACRYPT. LNCS, vol. 6477, pp. 56–75. Springer (2010)
34. Hosoyamada, A., Sasaki, Y.: Finding hash collisions with quantum computers by using differential trails with smaller probability than birthday bound. In: EUROCRYPT (2). LNCS, vol. 12106, pp. 249–279. Springer (2020)
35. Hosoyamada, A., Sasaki, Y.: Quantum collision attacks on reduced SHA-256 and SHA-512. In: CRYPTO (1). LNCS, vol. 12825, pp. 616–646. Springer (2021)
36. Isobe, T.: A single-key attack on the full GOST block cipher. In: FSE. LNCS, vol. 6733, pp. 290–305. Springer (2011)
37. Isobe, T., Shibutani, K.: All subkeys recovery attack on block ciphers: Extending meet-in-the-middle approach. In: Selected Areas in Cryptography. LNCS, vol. 7707, pp. 202–221. Springer (2012)
38. Khovratovich, D., Rechberger, C., Savelieva, A.: Bicliques for preimages: Attacks on skein-512 and the SHA-2 family. In: FSE. LNCS, vol. 7549, pp. 244–263. Springer (2012)
39. Kölbl, S.: Putting wings on SPHINCS. In: PQCrypto. LNCS, vol. 10786, pp. 205–226. Springer (2018)
40. Kölbl, S., Lauridsen, M.M., Mendel, F., Rechberger, C.: Haraka v2 - efficient short-input hashing for post-quantum applications. IACR Trans. Symmetric Cryptol. **2016**(2), 1–29 (2016)
41. Merkle, R.C.: One way hash functions and DES. In: CRYPTO. LNCS, vol. 435, pp. 428–446. Springer (1989)
42. Ni, B., Dong, X., Jia, K., You, Q.: (Quantum) collision attacks on reduced Simpira v2. IACR Trans. Symmetric Cryptol. **2021**(2), 222–248 (2021)
43. Preneel, B., Govaerts, R., Vandewalle, J.: Hash functions based on block ciphers: A synthetic approach. In: CRYPTO. LNCS, vol. 773, pp. 368–378. Springer (1993)
44. Sasaki, Y.: Meet-in-the-middle preimage attacks on AES hashing modes and an application to Whirlpool. In: FSE. LNCS, vol. 6733, pp. 378–396. Springer (2011)
45. Schrottenloher, A., Stevens, M.: Simplified mitm modeling for permutations: New (quantum) attacks. Cryptology ePrint Archive, Report 2022/189 (2022)
46. Suzuki, T., Minematsu, K.: Improving the Generalized Feistel. In: FSE. LNCS, vol. 6147, pp. 19–39. Springer (2010)
47. Zhang, G., Liu, M.: A distinguisher on PRESENT-like permutations with application to SPONGENT. Sci. China Inf. Sci. **60**(7), 72101 (2017)
48. Zheng, Y., Matsumoto, T., Imai, H.: On the construction of block ciphers provably secure and not relying on any unproved hypotheses. In: CRYPTO. LNCS, vol. 435, pp. 461–480. Springer (1989)