IACR Transactions on Symmetric Cryptology
ISSN 2519-173X, Vol. 2023, No. 3, pp. 146–183. DOI:10.46586/tosc.v2023.i3.146-183

# Simplified Modeling of MITM Attacks for Block Ciphers: New (Quantum) Attacks

André Schrottenloher[1]* and Marc Stevens[2]

[1] Univ Rennes, Inria, Centre National de la Recherche Scientifique (CNRS), Institut de Recherche en Informatique et Systèmes Aléatoires (IRISA), Rennes, France
firstname.lastname@inria.fr
[2] Cryptology Group, Centrum Wiskunde Informatica (CWI), Amsterdam, The Netherlands
firstname.lastname@cwi.nl

**Abstract.** The meet-in-the-middle (MITM) technique has led to many key-recovery attacks on block ciphers and preimage attacks on hash functions. Nowadays, cryptographers use automatic tools that reduce the search of MITM attacks to an optimization problem. Bao et al. (EUROCRYPT 2021) introduced a low-level modeling based on Mixed Integer Linear Programming (MILP) for MITM attacks on hash functions, which was extended to key-recovery attacks by Dong et al. (CRYPTO 2021). However, the modeling only covers AES-like designs. Schrottenloher and Stevens (CRYPTO 2022) proposed a different approach aiming at higher-level simplified models. However, this modeling was limited to cryptographic permutations.
In this paper, we extend the latter simplified modeling to also cover block ciphers with simple key schedules. The resulting modeling enables us to target a large array of primitives, typically lightweight SPN ciphers where the key schedule has a slow diffusion, or none at all. We give several applications such as full breaks of the PIPO-256 and FUTURE block ciphers, and reduced-round classical and quantum attacks on SATURNIN-Hash.

**Keywords:** MITM Attacks · Key-recovery attacks · Quantum cryptanalysis · Preimage attacks · AES · Present

## 1 Introduction

Meet-in-the-middle (MITM) attacks are powerful attacks on symmetric primitives such as block ciphers [DH77] and hash functions [Sas11]. The goal in both cases is to find an unknown value (the internal state or the key) which satisfies a certain set of constraints, e.g., between the inputs and outputs of a certain computational path.

If one guesses the entire unknown value, then the whole path can be recomputed and the constraints checked: this is the generic exhaustive search, which can always be applied. The MITM technique identifies some parts of the path which can be computed independently from others. Typically one defines a *forward chunk* (computed forwards) and an independent *backward chunk* (computed backwards), which start from a given *starting point*, and meet at several *matching points*. The values that they can take are computed independently. Afterwards, one selects among the pairs all those which satisfy the conditions set by the matching points.

MITM attacks have been successfully applied to many designs, using more and more refined techniques: guess-and-determine [DSP07], splice-and-cut [AS08, GLRW10], bicliques [KRS12], 3-subset MITM [BR10, Sas18], and so on. Even when they do not reach the best

*Part of this work was done while the author was at CWI.

Licensed under Creative Commons License CC-BY 4.0.
Received: 2023-06-01 Accepted: 2023-08-01 Published: 2023-09-19

number of rounds in terms of key-recovery, they can have other advantages, such as a very low data complexity (like the attack on Present-80 of [CNV13]).

**Automatic Search of MITM Attacks.** A MITM attack is entirely defined by its *characteristic* or *path*, the choice of its forward and backward chunks. Finding the best characteristic is an optimization problem, which can be solved automatically.

Dedicated search tools were proposed, such as the tool of Derbez and Fouque [DF16] for MITM and DS-MITM attacks. The algorithm starts from a given starting point, tries to expand the forward and backward paths before they match, and terminates if it has explored too many possibilities. Another automatic search, with a similar spirit, was proposed in [AA20]. While both these tools apply to a large class of designs (AES-like, bit-based SPNs, Feistel schemes), they have some limitations: the tool of [DF16] is not adapted for preimage attacks and the one of [AA20] does not support guess-and-determine. In a similar context, Hadipour and Eichlseder introduced a generic tool [HE22] for guess-and-determine attacks, but it does not cover MITM attacks.

In this paper, we focus on a *modeling* of the problem using Mixed Integer Linear Programming (MILP). Sasaki first used MILP to optimize the 3-subset MITM attack on Gift-64 [Sas18], however his modeling only concerned this specific case. Later, Bao et al. defined an MILP modeling of MITM attacks on AES-like hash functions [BDG+21]. The modeling relies on local *propagation rules* for the backward and forward paths, which constrain the state of nibbles (forward, backward, unknown. . . ) and count the number of degrees of freedom which are consumed. Later, it was extended to key-recovery attacks in [DHS+21], and to the guess-and-determine technique in [BGST22], including more and more techniques (nonlinearly constrained words, *superposition states*) which allowed to capture more refined constraints on the key nibbles and interactions between the key and the state.

Originally, this modeling strategy applies to designs which perform operations at a nibble level (though we note that bit-level models were also introduced afterwards in [QHD+23]). Besides, the definition of rules can become quite complicated. A different approach was proposed in [SS22a], which covers both AES-like and Present-like designs, and benefits from a simple description. However, the scope of the modeling is now limited to a computational path which does not take into account key additions. Quantum attacks are obtained by adapting the objective function of the search.

**Contribution and Results.** In this paper, we extend the modeling strategy of [SS22a] to support key additions in the primitive. The main issue posed by key additions is that they happen at the nibble level, while the model of [SS22a] considered larger components in the primitive. We show that we can keep the simplicity of the previous model by adding variables for the key nibbles, and placing them in the forward or backward chunks using a few new constraints. This allows us to cover both pseudo-preimage attacks on hash functions and key-recovery attacks on block ciphers. The downside is that we need to focus on *lightweight* key schedules such as the one of Present [BKL+07] which only perform bit rotations and S-Box operations.

Complex linear functions in the key schedule may complicate the interactions between the key and state bits. The series of works on AES-like designs [BDG+21, DHS+21, BGST22] have been able to capture more and more of these interactions, and reach new attacks that the previous models were not able to find. We conjecture that our modeling, despite its simplicity, reaches at least the same performance on the class of designs defined in Section 2.1. Though we cannot prove it in full generality, we could for example find a pseudo-preimage attack on 7-round Saturnin-Hash with a smaller time complexity than [DHS+21].

The results obtained with our tool are summarized in Table 1, compared with some results of previous works. We obtain full-round attacks on the block ciphers Future [GPS22] and Pipo-256 [KJK+20], both due to the simple key schedules of the ciphers. We also give new quantum attacks on reduced-round Saturnin-Hash and Saturnin, and observe that some Grover-meet-Simon attacks [LM17] are covered by the automatic search. This complements the automated search of Simon-based attacks previously proposed by Canale, Leander and Stennes [CLS22]. Though their tool can find more attacks than ours, as it allows to define periodic functions from a bigger class, it was limited to rather simple constructions. Our tool is more adapted to complex cipher designs.

**Table 1:** Attacks overview. Complexities are given in $\log_2$. The 'time' column includes the generic attack time. In the 'memory' column, '*' means QRACM and '**' means QRAQM.

| Target | Rounds | Time | Memory | Data | Reference |
|---|---|---|---|---|---|
| Classical | | | | | |
| 1k-AES | 7 | 112 / 128 | 40 | 80 | Section C.1 |
| 1k-AES | 7 | 120 / 128 | 32 | 32 | Section C.1 |
| 2k-AES | 10 | 248 / 256 | 48 | 128 | Section C.1 |
| Saturnin pre. | 7 / 16 | 208 / 256 | 48 | | [DHS+21] |
| Saturnin pre. | 7 / 16 | 192 / 256 | 160 | | Section 4.1 |
| Future | 10 / 10 | 126 / 128 | 34 | 64 | Section 4.2 |
| Saturnin | 7.5 / 10 | 244 / 256 | 244 | 225 | [CDL+20] |
| Saturnin | 6.5 / 10 | 152 / 256 | 70 | 70 | [FKL+00] (C.3) |
| Saturnin | 6.5 / 10 | 248 / 256 | 24 | 248 | Section C.3 |
| Haraka-512 | 5.5 / 5 | 240 / 256 | 16 | | [SS22a] |
| Haraka-512 | 5.5 / 5 | 224 / 256 | 80 | | Section C.4 |
| Haraka-512 | 6.5 / 5 | 240 / 256 | 224 | | Section C.4 |
| Present-80 | 7 / 31 | 73.42 / 80 | 9 | 1 | [CNV13] |
| Present-80 | 8 / 31 | 73.42 / 80 | | 6 | [CNV13] |
| Present-80 | 9 / 31 | 78 / 80 | 6 | 8 | Section 5.1 |
| Present-80 | 9 / 31 | 77 / 80 | 16 | 12 | Section 5.1 |
| Fly | 11 / 20 | 125 / 128 | 20 | 64 | Section 5.2 |
| Pipo-128 | 10 / 14 | 125 / 128 | 20 | 64 | Section 5.2 |
| Pipo-256 | 18 / 18 | 252 / 256 | 60 | 64 | Section 5.2 |
| Future | 10 / 10 | 126 / 128 | 34 | 64 | Section 4.2 |
| Quantum | | | | | |
| Saturnin pre. | 7 / 16 | 115.55 / 128 | 32** | | Section 4.1 |
| Saturnin | 6.5 / 10 | 120 / 128 | 70** | 70 (Q1) | [BNS19] (C.3) |
| Saturnin | 6.5 / 10 | 127.55 / 128 | 24** | 124 (Q2) | Section C.3 |
| Gift-64 (GMS) | 15 / 28 | 58.97 / 64 | negl. | 58.97 (Q2) | Section C.5 |
| Saturnin (GMS) | 5.5 / 10 | 110.24 / 128 | 64* | 64 (Q1) | Section C.3 |

**Outline.** The paper is organized as follows. In Section 2, we define the AES-like and Present-like classes of designs, introduce the MITM framework and the different techniques relevant for this paper, as well as quantum computing tools. We borrow from [SS22a] the generic conversion of a classical MITM attack into a quantum one, and we explain the Grover-meet-Simon (GMS) attacks.

Next, in Section 3 we explain the modeling of MITM attacks, first explaining [SS22a], then introducing our new variables and constraints for the key schedule and details on how to adapt the framework to the key-recovery case. The last sections are devoted to our applications: 1) AES-like designs in Section 4, including Saturnin-Hash, Saturnin and

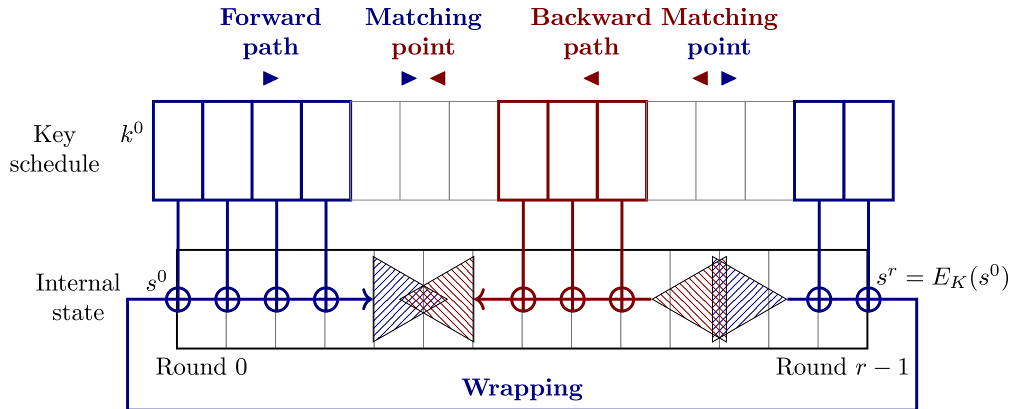


**Figure 1:** MITM attacks using the "splice-and-cut" technique (figure adapted from [SS22a]).

Future; 2) Present-like designs in Section 5, including Present and Pipo.

The code used in this paper, including the generation of MILP models for all our examples, and the generation of figures, is available online[1]. We used the Gurobi solver [Gur23] on a desktop computer.

## 2 Preliminaries

In this section, we give a high-level overview of MITM attacks, with corresponding formulas for the time complexity. We also give preliminaries on quantum attacks.

As shown in Figure 1, we consider a block cipher $E_K$, made of a sequence of $r$ rounds ($0$ to $r-1$) which act on the *internal state* taking value $s^0$ (plaintext) to $s^r$ (ciphertext). A *key schedule* function transforms the master key $K$ into a sequence of round keys $k^0, \ldots, k^r$. Each $k^i$ is added before round $i$, and $k^r$ is the final key addition. We enforce some relation between $s^0$ and $s^r$, which creates a *closed computational path*. We are searching for a pair $(s^0, K)$ satisfying this path.

In the case of a preimage or pseudo-preimage attack [Sas11], the block cipher $E_K$ is used as a compression function. For example, in the Matyas-Meyer-Oseas (MMO) mode, the function is $f(m, h) = E_h(m)$ for a chaining value $h$ and a message block $m$. Given a target $T$, a pair $h, m$ such that $E_h(m) = m \oplus T$ is a preimage of the compression function. The degrees of freedom of both $m$ and $h$ may be used. If we fix $h$, we remove the key schedule; the block cipher becomes a permutation.

For a key-recovery attack, the relation between input and output ("wrapping" in Figure 1) is given by $E_K$. On the forward path in Figure 1, we need to compute $s^0$ from $s^r$, so we perform decryption queries.

### 2.1 AES-like and Present-like Designs

The AES [DR20] is the standardized version of the block cipher Rijndael [DR99], and its design strategy has been followed by many ciphers and hash functions, forming a family that we will call "AES-like". AES-like designs are substitution-permutation networks (SPN) in which the internal states and round keys are represented as matrices of nibbles. The S-Box layers operate on nibbles independently. The linear layer can also be expressed as an operation on nibbles.

[1] https://github.com/AndreSchrottenloher/key-mitm

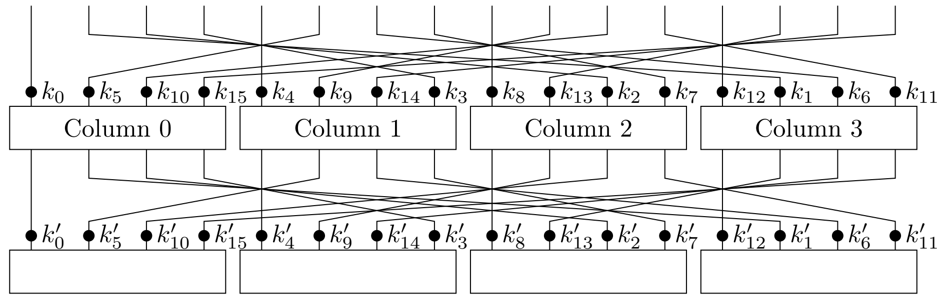


**Figure 2:** Two rounds of AES with two keys $k$ and $k'$, using the Super S-Box representation.

In the AES, the state is a $4 \times 4$ matrix of bytes, and the round function composed of: the key addition (ARK); the S-Box layer (SB); the ShiftRows operation (SR), which permutes bytes within rows of the matrix; and finally, the MixColumns operation (MC), which multiplies each column by an MDS matrix. Other AES-like designs typically have similar operations. Sometimes (like in Skinny [BJK+16]) the MixColumns is not MDS, but we will not consider such cases in this paper. Key schedules of AES-like designs also typically operate at the nibble level, e.g., in the AES itself, the only operations are XORs of key bytes and S-Boxes.

**Super S-Box and Present-like Ciphers.** It can be noticed that in the sequence of operations ARK $\rightarrow$ SB $\rightarrow$ SR $\rightarrow$ MC, the operation SB commutes with SR and ARK. Thus, an AES round can be rewritten as a permutation of bytes, followed by a key addition, and a layer of *Super S-Boxes* applying on columns. This abstraction is essential for us. Using the Super S-Box representation, an AES-like design becomes an SPN in which the linear layer is simply a permutation of nibbles, corresponding to the SR operation. We give an example in Figure 2, where the initial bytes are numbered from 0 to 15, and the columns of the state are the groups of bytes $[0-3], [4-7], [8-11], [12-15]$.

We name *Present-like* the SPN ciphers which use only a nibble permutation as their linear layer, like the block cipher Present [BKL+07]. Though Present uses a *bit* permutation, our structure considers nibbles of any size. Our modeling starts from this family of ciphers, and covers AES-like designs thanks to the Super S-Box representation.

**Key Schedules.** The key schedule of a cipher is an in-place operation that transforms a set of internal *key registers* and extracts rounds keys. All variants of AES have a key schedule which offers quite a strong diffusion. This renders the AES itself immune to MITM key-recovery attacks[2]. However, lightweight ciphers, whether AES-like (e.g., Saturnin [CDL+20]) or Present-like (e.g., Present, Gift [BPP+17]) can have much simpler key schedules. In this paper, we consider key schedules that operate nibble-wise using in-place operations, either: 1) a permutation of nibbles; or 2) an S-Box applied in place to some nibbles. This includes the key schedule of Present.

## 2.2 Overview of Techniques

Below we briefly review the techniques that will be contained in our automatic search. We consider both key-recovery and preimage attacks. While they are very different conceptually, the modeling is quite similar. Let $E_K$ be a block cipher with block length (in nibbles) $n$ and key length $|K|$. We start with the key-recovery case.

[2]Demirci-Selçuk Meet-in-the-middle attacks [DS08], which do not rely as much on the key schedule as the simple MITM, are sometimes named "MITM" in the literature. We do not consider them here.

We have query access to $E_K$ and wish to retrieve $K$ in time less than $2^{|K|}$ computations of $E$. This is done by finding solutions to the closed computational path represented in Figure 1.

We formalize this using a *forward path* function $f_F(x, k_F)$ and a *backward path* function $f_B(x, k_B)$ that take a common starting state $x$ (e.g., the plaintext), a forward (resp. backward) key guess $k_F$ of length $|k_F|$ (resp. $k_B$ of length $|k_B|$) and compute the same set of nibbles at some round inside the cipher.

We see that $k = k_F \cup k_B$ can be a potential candidate[3] for $K$ only if there exists $x$ such that $f_F(x, k_F) = f_B(x, k_B)$. Typically, starting from a constant $x$, we compute $E_K(x)$, compute forwards from $x$ and backwards from $E_K(x)$. Either $f_F$ or $f_B$ includes a query to $E_K$ or $E_K^{-1}$.

Let $|f| := |f_F| = |f_B|$ be the output size of $f_F$ and $f_B$. From now on, these sizes are counted *in nibbles*, and the complexity formulas would need to be re-scaled depending on the nibble size. The attack does:

1. fix the value $x$
2. compute the list of all $k_B, f_B(x, k_B)$: time $2^{|k_B|}$ (we neglect the time to sort the list by the second entry)
3. for each $k_F$, compute $f_F(x, k_F)$ and enumerate all the pairs $k_F, k_B$ such that $f_F(x, k_F) = f_B(x, k_B)$: time $2^{|k_F|} + 2^{|k_F \cup k_B| - |f|}$. For each pair, recompute the whole path and check if it matches.

Because we can swap the roles of forward and backward paths, the time complexity for key-recovery is $2^{|k_B|} + 2^{|k_F|} + 2^{|k_F \cup k_B| - |f|}$ and the memory complexity is $\min(2^{|k_B|}, 2^{|k_F|})$.

**Varying the Cut Set / Initial Structure.** The *cut set* [AAMA14] or more generally, the *initial structure*, is the start of the path, i.e., the starting state $x$ defined above. It can take any value and *any position* within the closed path. It can also be scattered onto multiple rounds. If the cut set is moved inside the closed path, the query complexity of the MITM attack, which was initially 1, increases. Indeed, each computation of $f_F$ (resp. $f_B$) will potentially query $E_K$ on a new value, depending on the starting state $x$ *and* on the current key guess $k_F$ (resp. $k_B$).

**3-Subset MITM Attack.** The 3-subset MITM attack [BR10] partitions the key space in three sets. Alongside the forward and backward key nibbles, we define *shared* key nibbles $k_S$ of length $|k_S|$, and the forward and backward paths are now computed as: $f_F(x, k_F, k_S)$ and $f_B(x, k_B, k_S)$, where $k_F$ and $k_B$ are disjoint. The MITM attack now runs as follows: we fix $x$, and for each choice of $k_S$, we compute the pairs $(k_F, k_B)$ such that $f_F(x, k_F, k_S) = f_B(x, k_B, k_S)$. The time complexity is $2^{|k_S|+|k_B|} + 2^{|k_S|+|k_F|} + 2^{|k_F|+|k_B|+|k_S|-|f|}$ and the memory complexity is $\min(2^{|k_B|}, 2^{|k_F|})$.

**Guess-and-Determine.** Guess-and-determine (GAD) is a powerful technique both for key-recovery [BDF11] and preimage attacks [SWWW12, BGST22]. We introduce *nibble state guesses* $y_F$ and $y_B$ in the definition of the forward and backward paths: $f_F(x, y_F, k_F, k_S)$ and $f_B(x, y_B, k_B, k_S)$. We then have to compute $f_F$ and $f_B$ not only for all $k_F$ and $k_B$, but also $y_F$ and $y_B$, and find quadruples $(y_F, y_B, k_F, k_B)$ such that $f_F(x, y_F, k_F, k_S) = f_B(x, y_B, k_B, k_S)$. The time complexity becomes:

$$2^{|k_S|+|k_B|+|y_B|} + 2^{|k_S|+|k_F|+|y_F|} + 2^{|k_S|+|k_B|+|y_B|+|k_F|+|y_F|-|f|} \quad , \tag{1}$$

[3] In this paper, the forward, backward (and shared) key nibbles will entirely cover the master key. It is not necessarily the case, but it happens when the key-schedule is very simple.

and the memory complexity is $\min(2^{|k_B|+|y_B|}, 2^{|k_F|+|y_F|})$. To have a valid attack, we need $|y_B| + |y_F| < |f|$, i.e., the guesses allow to deduce more matching state nibbles.

Finally, we note that we can make $x$ bigger than $n$ nibbles, i.e., fix more than a full internal state. In that case we will need to repeat the merging operation $|x| - n$ times before finding a valid pair of internal state and key. The time complexity becomes:

$$2^{|k_S|} \times 2^{\max(|x|-n,0)} \times \left(2^{|k_B|+|y_B|} + 2^{|k_F|+|y_F|} + 2^{|k_B|+|y_B|+|k_F|+|y_F|-|f|}\right) \quad . \tag{2}$$

**Preimage Attack.** In the preimage case, we want to find a pair $(x, K)$ such that $E_K(x) \oplus x = T$ for a fixed target $T$, in time less than $2^n$ computations of $E$. Informally, the problem remains the same: having fixed $k_S$ and $x$, find the pairs $(y_F, k_F), (y_B, k_B)$ such that $f_F(x, y_F, k_F, k_S) = f_B(x, y_B, k_B, k_S)$, among which a solution will be found. The main difference is the number of times that we need to repeat this operation.

If the path contains only one solution, and if the key is entirely fixed, we can see that we will have to repeat $2^{|x|}$ times to traverse all the possible internal states, and find the one that matches. If there are $2^t$ solutions (due to less "wrapping" between the input and output), then we need only $2^{|x|-t}$ repetitions.

The forward and backward key nibbles take $2^{k_B+k_F}$ different values. Assuming that changing these key nibbles re-randomizes properly the path, this means that we only have to repeat $2^{|x|-t-k_B-k_F}$ times to traverse $2^{|x|-t}$ different starting values for $x$. This gives the formula:

$$2^{\max(|x|-|k_B|-|k_F|-t,0)} \times \left(2^{|k_B|+|y_B|} + 2^{|k_F|+|y_F|} + 2^{|k_B|+|y_B|+|k_F|+|y_F|-|f|}\right) \quad . \tag{3}$$

**All-subkey-recovery (ASR) and Parallel MITM (PMITM).** The ASR technique was proposed in [IS12]. If the key schedule is complex, we cannot use the relations between round keys, and the effective key length increases. Any guess $k_F$ yields a value $f_F(x, k_F, k_S)$ for the matching variable, and any guess $k_B$ yields a value $f_B(x, k_B, k_S)$. However, the amount of matching may be too small and leave more than $2^{|K|}$ triples $(k_F, k_B, k_S)$. The *Parallel MITM* technique overcomes this by increasing the effective output length of $f_F$ and $f_B$: it computes in parallel for multiple starting states $x$. The time and memory complexities are increased by a small factor with respect to a single starting state.

The PMITM is not very useful in our case, where the key schedule is simple. Furthermore, it is incompatible with GAD. Indeed, the internal state guesses $y_F, y_B$ are dependent on the starting state $x$. Multiple parallel states would have multiple independent state guesses, and the complexity would increase faster than it decreases thanks to the PMITM.

**Sieve-in-the-middle (SIM) and Bicliques.** The sieve-in-the-middle technique [CNV13] allows to sieve through an additional S-Box layer between the forward and backward path. This is done by means of an advanced list merging procedure. Sieve-in-the-middle is *partially* captured by GAD: though the merging procedure in the SIM attacks is more complex, it can be captured by making guesses on the forward or on the backward side. However, the modeling will not capture the fact that these guesses cost less than a recomputation of the full subpath, since they are made at the last moment.

Biclique cryptanalysis is an extension of MITM attacks used both against block ciphers [BKR11] and hash functions [KRS12], where the initial structure is extended with the help of bicliques. We did not study bicliques in this paper. It is still an open question to extend the MILP models of MITM attacks with bicliques.

## 2.3 Quantum Computing

We assume basic knowledge of quantum computing in the quantum circuit model, such as the ket notation $|\cdot\rangle$, qubits and basic operations (see, e.g., [NC02]). We use "time" to refer

to gate counts in quantum circuits, and "quantum memory" for their width (number of qubits), in multiple of the time and memory required to implement the primitive attacked. We stress that we use only generic complexity formulas, and no technical knowledge of quantum computing is required to apply these formulas.

**Quantum Memories.** We use in this paper two types of *quantum memory*, which can be understood as a hardware assumption: QRACM (quantum-accessible classical memory) and QRAQM (quantum-accessible quantum memory). QRACM allows unit-time access *in superposition* to a large classical memory. That is, given memory cells $y_0, \ldots, y_{M-1}$, the operation: $|i\rangle |0\rangle \mapsto |i\rangle |y_i\rangle$ (memory access) can be implemented in time 1. Whereas QRAQM is a stronger model in which the following operation costs a time 1: $|i\rangle |y_0, \ldots, y_{M-1}\rangle |x\rangle \mapsto |i\rangle |y_0, \ldots, y_{i-1}, x, y_{i+1}, \ldots, y_{M-1}\rangle |y\rangle$, which allows at the same time to read and write in superposition in the available quantum memory.

As can be seen in Table 1, our quantum MITM attacks require QRAQM, except Grover-meet-Simon (key-recovery) attacks.

**Grover's Algorithm.** Grover's search algorithm [Gro96] solves the following problem: given a quantum oracle that implements a boolean function $f : \{0,1\}^n \to \{0,1\}$, find a preimage of 1. It is an iterative procedure that moves its internal state towards the uniform superposition of solutions. If there are $t$ such preimages, then $\frac{\pi}{4}\sqrt{\frac{2^n}{t}}$ calls to the oracle $O_f : |x\rangle |0\rangle \to |x\rangle |f(x)\rangle$ are sufficient to reach a state very close to: $\frac{1}{\sqrt{t}} \sum_{x, f(x)=1} |x\rangle$. Measuring this state allows to find a solution. For example, exhaustive key search uses a few plaintext-ciphertext pairs $(p_i, c_i)$ and tests: $f(k) = 1 \iff \forall (p_i, c_i), E_k(p_i) = c_i$. With 1 expected solution, only $\mathcal{O}(2^{|k|/2})$ calls to a quantum implementation of $E_k$ are required, instead of $\mathcal{O}(2^{|k|})$ classically (a quadratic speedup).

The calls to $O_f$ usually dominate the cost of the algorithm. Implementing the function $f$ might require the QRACM or QRAQM model, for example when we need to access a large list.

**Quantum Queries: the Q1 and Q2 model.** The literature on quantum key-recovery attacks (e.g., [KLLN16b]) distinguishes two quantum attacker settings: Q1 (classical queries) and Q2 (quantum queries). In Q1, the quantum attacker has only access to classical secret-key oracles (decryption and / or encryption), whereas in Q2, the quantum attacker can use a *quantum oracle*, for example: $O_{E_K} : |x\rangle |0\rangle \mapsto |x\rangle |E_K(x)\rangle$. In particular, any computation that involves the secret-key cipher $E_K$ can now be done in superposition.

If the key length is twice the block size or more, then we can use QRACM to emulate the Q2 setting. Indeed, the cost of querying the entire codebook (classically) is lower than the cost of Grover search for the key. Thus, any quantum attacker can pre-compute the codebook, store it in QRACM, and implement $O_{E_K}$ efficiently.

This relation between Q2 queries and QRACM will also dictate our benchmarking of the latter: while counting the time in block cipher evaluations, we will consider a Q2 query to cost 1, and a QRACM / QRAQM query to cost 1 as well.

## 2.4 Quantum Attacks

We reuse Theorem 4 from [SS22b, Appendix A.2], which converts a classical MITM into a quantum attack using quantum search (the exact variant of Amplitude Amplification [BHMT02]). The quantum attack may benefit from up to a quadratic speedup with respect to the classical attack.

It considers a MITM attack that first makes $g$ guesses, then merges two lists of size $2^{\ell_F}$ and $2^{\ell_B}$ into a list of size $2^{\ell_M}$. It also assumes heuristically that there is a single solution, and that the subpaths leading to the solution are selected u.a.r. from all choices.

**Theorem 1.** *Let $T$ be the quantum time to compute an element of any of the three lists, or to recompute the full path given a partial match. There is a quantum algorithm of complexity:*

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

*that finds the key $K$ with probability* $1/2$.

Intuitively, the algorithm uses a Grover search on the $g$ guesses, under which a two-list merging is used. The two-list merging constructs the forward list first, then performs a quantum search in the backward and merged list elements. Note that the roles of forward and backward can be exchanged in this formula. We refer to [SS22b] for the details of this algorithm and its correctness.

In the following, like in [SS22a], we consider that $T$ is smaller or equal to the quantum time necessary to implement the full primitive under attack, and we count the time in multiples of $T$. This simplifies the comparison with Grover search.

**Q1 and Q2 Model.** Though the procedure of [SS22a] remains valid for key-recovery attacks, the computation of one of the two lists now includes queries to the block cipher (or its inverse). For example, assume that we access the block cipher $E$ itself. Let $S$ be the space of plaintexts encountered during the attack. Often $S$ becomes the full codebook, but for exhaustive search (which can be seen as a degenerate MITM attack), $S$ contains only a constant number of elements. Let $E' : S \to \{0,1\}^n$ be the cipher restricted to the space $S$.

In the classical attack, we compute the lists using classical access to $E'$. In the quantum attack, the list elements are computed in superposition. This means that we need superposition query access to $E'$. If $S$ is small enough with respect to the total key size, this can be done via QRACM (sometimes at the expense of a bigger memory) in the Q1 model. Otherwise, this requires the Q2 model.

## 2.5 Grover-meet-Simon as a PMITM Attack

Simon's algorithm [Sim97] is a quantum algorithm that solves the *boolean hidden period* problem: given access to a two-to-one function $f : \{0,1\}^n \to \{0,1\}^m$ such that $f(x \oplus s) = f(x)$, for some secret $s$, find $s$. The algorithm runs in time $\mathcal{O}(n^3)$ using $\mathcal{O}(n)$ Q2 queries to $f$. It has been analyzed in detail in multiple works (e.g. [KLLN16a, SS17]) and found to be quite robust, in particular it works well if $f$ is a *random periodic function* [Bon21], which models well the cases encountered in symmetric cryptography.

Simon's algorithm is the key component of many quantum attacks in the Q2 setting [KLLN16a] and more recently the Q1 setting [BHN+19]. In this paper, we will focus on the Grover-meet-Simon (GMS) attack of Leander and May [LM17]. This is a Q2 attack that *finds* a periodic function in a family of functions:

$$\begin{cases} F : \{0,1\}^\kappa \times \{0,1\}^n \to \{0,1\}^m \\ \forall z, F_z = F(z,\cdot) \\ \exists! k \in \{0,1\}^\kappa, (\exists s, \forall x, F_k(x) = F_k(x \oplus s)) \end{cases} .$$

GMS uses a Grover search of the right $k$, which calls Simon's algorithm as a test. Its quantum complexity has been analyzed in detail in [Bon21, BJ22]. As for Simon's algorithm, the constant in the $\mathcal{O}(n)$ is close to 1 if we assume the functions to be random, and the gate count in $\mathcal{O}(n^3)$ is usually smaller than the cost of $n$ function computations.

Even if we restrict the functions' output to one bit, this only doubles the number of queries needed by Simon's algorithm [MS22]. Therefore, we will approximate the complexity as:

$$4T\frac{\pi}{4}2^{|k|/2}n \ , \tag{5}$$

where $T$ is the cost of a query to $F$.

**GMS-based MITM Attacks.** Consider a PMITM attack. Starting from an initial state $x$, it looks for a collision between the forward and backward path: $(k_F, k_B, k_S)$ such that $f_F(x, k_F, k_S) = f_B(x, k_B, k_S)$. The functions $f_F$ and $f_B$ can have a single-bit output.

Assume that both $f_F$ and $f_B$ are of the form:

$$f_F(x, k_F, k_S) = g_F(x \oplus k'_F, k_S) \text{ and } f_B(x, k_B, k_S) = g_B(x \oplus k'_B, k_S)$$

where $k'_F$ and $k'_B$, of size $|x|$, contain nibbles of $k_F$ and $k_B$ respectively (and zero nibbles for positions at which no key is added). We are then looking for $(k_S, k'_B, k'_F)$ such that:

$$\forall x, g_F(x, k_S) = g_B(x \oplus k'_F \oplus k'_B, k_S) \ .$$

Therefore, when looking for MITM attacks, we can also look for GMS-based attacks. In this case, we will remove the GAD technique. We separate the key nibbles into "shared" key nibbles $k_S$ (guessed with Grover search) and "middle" key nibbles $k_M$ (found with Simon's algorithm). The part of the input $x$ on which there is no addition of $k_M$ is fixed.

Though we would have liked to use the offline-Simon algorithm [BHN+19], which removes the need for Q2 queries or QRACM, a technical problem arises from the fact that the functions computing the forward and backward paths are not invertible.

# 3 Modeling

In this section, we describe the MILP modeling of MITM attacks used in this paper. It relies on a *cell-based* representation of a cipher, which is the same as in [SS22a].

## 3.1 Previous Work

An $r$-round Present-like cipher is abstracted as an $r$-layered, weighted, undirected graph. The nodes of the graph, named *cells*, correspond to the S-Box operations in the cipher. The edges of the graph correspond to the exchange of nibbles in the linear layer. The MITM input-output constraint that closes the path is enforced by edges between cells of layer 0 (first round) and layer $r-1$ (last round). Intuitively, this corresponds to the rewriting of the MITM problem as a system of linear relations between the cells, since all the nonlinear operations are local to them. Both cells and edges are weighted. For any pair $(c, c')$, we use $w_c$ and $w_{c,c'}$ to denote these weights (we have $w_{c,c'} = 0$ if $(c, c')$ is not an edge). They correspond to the number of state nibbles that one should know to determine the value of this cell or edge. For example, in the graph representation of the AES, one has $w_{c,c'} = 1$ for edges and $w_c = 4$ for 4-nibble cells.

Given any set of cells $C$, we define the *reduced list* $\mathcal{R}[C]$ which contains all assignments of values to these cells satisfying the linear constraints. It has size:

$$\log_2 |\mathcal{R}[C]| = \sum_{c \in C} w_c - \sum_{(c,c') \in C^2} w_{c,c'} \ . \tag{6}$$

Thanks to the Present-like structure, as long as $C$ does not cover all the rounds, there is a streaming algorithm that produces the elements of $\mathcal{R}[C]$ in time $|\mathcal{R}[C]|$. Indeed, incoming

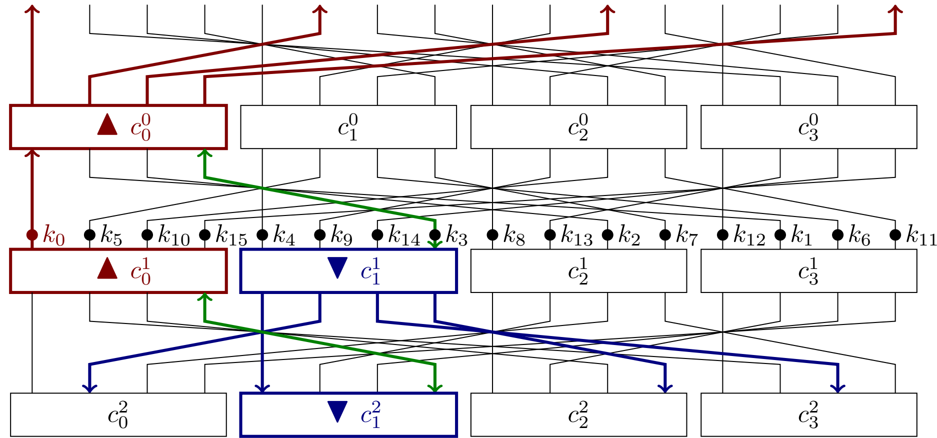


**Figure 3:** Two global guesses (**green** $\leftrightarrow$ edges) in the path of an AES-like cipher.

edges of each cell are either fixed, determined by the previous round, or undetermined. Then they correspond to forward or backward guesses $y_F, y_B$, and can be arbitrarily filled.

The MITM configuration is now defined using two sets of cells: *forward* ($C_F$) and *backward* ($C_B$), such that $C_M := C_F \cup C_B$ contains a whole round. As shown in Equation 6, the respective sizes $\ell_F, \ell_B, \ell_M$ of $\mathcal{R}[C_F]$, $\mathcal{R}[C_B]$, $\mathcal{R}[C_M]$ (in $\log_2$) can be deduced from the choice of $C_F, C_B$ by linear inequations.

To any choice of $C_F, C_B$, there corresponds a MITM algorithm of memory $\min(\ell_F, \ell_B)$ and running time $\max(\ell_B, \ell_F, \ell_M)$ (in $\log_2$), which consists in constructing one list, streaming the other and re-computing the path for matching pairs.

**Global Guesses.** Any edge $(c, c')$ where $c \in C_B, c' \in C_F$ can be turned into a "global guess", as shown in Figure 3. Indeed, by guessing these values before computing $\mathcal{R}[C_F]$ and $\mathcal{R}[C_B]$, the matching on these nibbles is precomputed. If there are $g$ nibbles of global guess, the time complexity becomes: $g + \max(\ell_B, \ell_F, \ell_M)$ (in $\log_2$). The quantum time complexity is obtained by simplifying Equation 4:

$$\frac{g}{2} + \max\left(\min(\ell_F, \ell_B), \frac{1}{2}\max(\ell_F, \ell_B, \ell_M)\right) . \tag{7}$$

Forward cells are represented in **blue** ▼, backward cells in **red** ▲. Global guesses are represented in **green** $\leftrightarrow$. It should be noted that they only impact the memory complexity of the attack and its quantum time complexity.

**Matching through MixColumns.** The difference between the PRESENT-like and AES-like cases lies in the following property of the Super S-Box, due to the MDS property of MixColumns. Let $c$ be a Super S-Box of width $w_c$.

Assume that we know $f, b$ nibbles at the *previous round*, from the forward and backward path, respectively, and $f', b'$ nibbles at the *next round*. Then we can match a total of $f + b + f' + b' - w_c$ nibbles *through* the cell, i.e., even if it does not belong to $C_F \cup C_B$. This is because we can write linear relations between these values. This is modeled by allowing to add such a cell in the merged list (we call it a *new merged cell*).

We can also *precompute* some of these relations, like global guesses of nibbles. With limitations: • we cannot precompute more than $f + b + f' + b' - w_c$ relations (the total amount of matching on this cell); • we cannot precompute more than $f'$ (the amount of forward nibbles at the next round) and more than $b$ (the amount of backward nibbles at the previous round), otherwise this would create too many constraints.

**Modeling.** For each cell $c$, we create 3 boolean variables $\mathsf{cell_col}_L[c]$ where $L \in \{F, B, M\}$. We always have $\mathsf{cell_col}_F[c] + \mathsf{cell_col}_B[c] \leq \mathsf{cell_col}_M[c]$, and this is an equality if the cell is not a Super S-Box.

The list sizes and precomputed matchings are deduced from these variables using linear inequalities. For each cell, we sum its *contribution* to the list size (as a new temporary variable), and the amount of global reduction that it allows. We deduce the time complexity (in $\log_2$), which is minimized. A complete pseudocode of this model is given in Appendix A.

## 3.2 Modeling the Key Schedule

Now, we move further than [SS22a]. Including a key schedule path is a simple transformation to the graph, where each edge is now (possibly, but not necessarily) labeled by a key nibble, as in Figure 3. This is the key nibble that is XORed on this edge.

Let us start by the new variables and constraints related to the key schedule itself. We consider a key schedule like the one of Present, which starts by initializing a *key register*, and then: • creates new key nibbles by applying operations in place on the key register; • extracts the round keys from the key register.

In a similar spirit as the cell-based modeling of the state update function, we start by considering all round key nibbles as independent. We assume that they all have the same size (e.g., a single bit for Present). For example, each round of Present-80 applies a single 4-bit S-Box on the key register, so after $r$ rounds, the total amount of key nibbles is $80 + 4r$.

The operations that were applied create *relations* between groups of key nibbles, e.g. $k_4|k_5|k_6|k_7 = S(k_0|k_1|k_2|k_3)$. These relations can be modeled as *key cells*, on the same principle as state cells. For a key cell of width $w$ (here 4), the knowledge of at least $w$ key nibbles allows to deduce all others.

**Modeling.** For each key nibble $k_i$, we create 3 boolean variables $\mathsf{key_col}_L[k_i]$ for $L \in \{S, B, F\}$. The key nibble can be either: • forward (F), i.e., known only in the forward path; • backward (B), i.e., known only in the backward path; • shared (S), i.e., known in both paths. This corresponds to $k_F, k_B, k_S$ in the formulas of Section 2.2.

For each key cell $c$, we create 3 exclusive boolean variables: $\mathsf{key_cell_col}_L[c], L \in \{S, B, F\}$, which indicate whether the key cell is "active" in the list of shared, forward or backward key nibbles. These variables have the following constraints:

- if $\mathsf{key_cell_col}_S[c]$ is true, then all key nibbles attached to this cell must be shared
- if $\mathsf{key_cell_col}_B[c]$ is true, then all key nibbles must be shared or backward
- if $\mathsf{key_cell_col}_F[c]$ is true, then all key nibbles must be shared or forward

The "activation" of a key cell means that we use its relation to reduce the number of effective key nibbles, either in the shared, the backward or forward sets. Thus we compute the number of effective shared, forward and backward key nibbles by a sum over all key nibble variables, minus the reduction of key cells.

## 3.3 Modeling the Key Addition

We observe that the key addition can be modeled using only *constraints* on the coloration of key nibbles. For any key nibble $k_i$, consider any edge $e = (c, c', w_{c,c'}, k_i)$ where $k_i$ is added:

- If both $c$ and $c'$ are in the forward path, then $k_i$ should be either forward, or shared. Indeed, in order to compute the next forward cell, we have the choice between guessing $k_i$ or guessing the state nibble. This costs the same, but guessing $k_i$ will

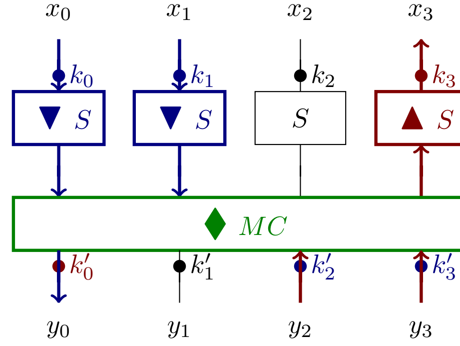


**Figure 4:** Key addition in the Super S-Box case.

always be more advantageous, since we might reuse this guess somewhere else in the path.

- Likewise, if both $c$ and $c'$ are in the backward path, then $k_i$ should be either backward, or shared.

While it is not necessary for all the key nibbles to be colored, e.g., in the PMITM case, in practice this will be the case in our attacks, since we consider simple key schedules. So the forward-backward (matching) and backward-forward (global guess) relations do not create any constraint.

**Computation of the Lists.** The forward and backward lists now contain both partial states and key guesses. We will start in both cases by taking values for the key nibbles, using the key cells, then compute the states round by round as before. The constraint of having at least one round not covered by $C$ still allows to enumerate the elements of $\mathcal{R}[C]$ efficiently.

The list sizes are computed as before (by the contribution of each cell individually), then increased by the amount of key nibbles that they contain. In the preimage case, the time complexity reflects Equation 3. In the key-recovery case, the time complexity is given by Equation 2. If there are $|x|$ nibbles of global reduction, and if the key-less list sizes were $\ell_B, \ell_F, \ell_M$, then the times are respectively, in $\log_2$:

$$\left.\begin{array}{c}\max(|x| - |k_B| - |k_F|, 0) \\ |k_S| + \max(|x| - n, 0)\end{array}\right\} + \max(|k_B| + \ell_B, |k_F| + \ell_F, |k_B| + |k_F| + \ell_M) \ . \tag{8}$$

In the quantum setting, these formulas are adapted using Equation 4:

$$\left.\begin{array}{l}\frac{1}{2}\max(|x| - |k_B| - |k_F|, 0) \\ \frac{1}{2}\left(|k_S| + \max(|x| - n, 0)\right)\end{array}\right\} + \max\Big(\min(|k_F| + \ell_F, |k_B| + \ell_B), \\ \frac{1}{2}\max(|k_B| + \ell_B, |k_F| + \ell_F, |k_B| + |k_F| + \ell_M)\Big) \ . \tag{9}$$

**Super S-Box Case.** To understand what happens if the cell $c$ is a Super S-Box, we must separate the S-Box layer and the MixColumns operation, as shown in Figure 4. Following the usual layout of SPN ciphers, the S-Box layer is always performed between the key addition and the MC operation.

In this example, there are 6 nibbles known in input and output, so we can form two linear relations between them, of the form: $L(S(k_0 \oplus x_0), S(k_1 \oplus x_1), S(k_3 \oplus x_3), y_0 \oplus k'_0, y_2 \oplus k'_2, y_3 \oplus k'_3) = 0$, where $L$ is a linear function. One of these relations can be precomputed

as follows: $L_F(S(k_0 \oplus x_0), S(k_1 \oplus x_1), y_0, k'_2, k'_3) = L_B(S(k_3 \oplus x_3), k'_0, y_2, y_3) = t$. That is, we can guess the value of $t$ beforehand. During the forward path computation, we will obtain $y_0$ as a function of $t$ and the other variables, and in the backward path, we obtain $x_3$ as a function of $t$ and the other variables.

To enable this, we just need to ensure the following additional constraints for each edge $(c, c', w_{c,c'}, k_i)$: • if $c$ is forward and $c'$ is a *new* merged cell, then $k_i$ must be forward; • if $c$ is backward and $c'$ is a new merged cell, then $k_i$ must be backward. These constraints allow to go through the upper S-Box layer and access the inputs to the MC layer. In particular, there are *no* constraints on the key nibbles below.

*Remark* 1 (Double key addition). In the path of a preimage attack of a compression function, we may have a key addition before the first round 0, and after the last round $r - 1$. This means that *two* key nibbles are added on the edges that connect rounds 0 and $r - 1$. Since our model does not allow two key additions on an edge, we add a new round of "dummy" single-nibble operations between the two additions.

### 3.4 Adaptation to the Key-recovery Setting

In the key-recovery case, going from round 0 to round $r - 1$ requires a query to the block cipher. This is modeled by adding a new round with a single cell (the "cipher cell"), as large as the internal state, which is connected to round 0 and to round $r - 1$. The cipher cell must belong either to $C_F$ (decryption queries) or to $C_B$ (encryption queries).

**Role of the Global Reduction.** The nibbles of global reduction serve as a way to reduce the memory complexity in preimage attacks. In key-recovery attacks, the same variables correspond to the cut-set. This change of perspective is visible in Equations 8 and 9. While a preimage attack requires to loop over the values given to the global guesses, a key-recovery attack can fix $n$ of these nibbles for free.

**Data Complexity.** In key-recovery attacks, we may want to control or minimize the data complexity. This can be done as follows. Assume without loss of generality that the cipher cell $c$ belongs to $C_B$. We look at all the edges connecting $c$ with cells at round 0, and count the weight of edges that: • connect with forward cells (thus, can be globally fixed); • do *not* have a backward key nibble. The data complexity can be upper bounded by the state size, minus the sum of weights of all such edges: this corresponds to the total number of state bits that can vary when the queries are made. Similar constraints apply if $c$ belongs to $C_F$.

**PMITM and Grover-meet-Simon.** In the PMITM case, we add the constraints that all cells should have a zero contribution to the list sizes (which will only depend on the initial state, and on key bit guesses), and we do not count the merged list size. We only require some matching between the forward and backward paths to occur.

Grover-meet-Simon is a subcase of PMITM, as we have seen in Section 2.5. In this case, we can compute an approximation of the complexity by counting only the "shared" key nibbles of the path. The key nibbles XORed to the starting state are not colored. They are the "middle" key nibbles $k_M$ defined in Section 2.5.

## 4 Application to AES-like Designs

In this section, we detail two of our applications to AES-like designs (more can be found in Appendix C). In each case, we only give the details of one solution path, but several solutions actually lead to the same complexity. How these paths translate into attack

algorithms was discussed previously in Section 2. For completeness, we include detailed descriptions of the attacks in Appendix B, and we only give here the main ideas.

The figures use the following color / symbol scheme: **green** ↔ for global guesses, shared key nibbles $k_S$ and ⧫ matching through MixColumns; **blue** ▼ for the forward path and $k_F$; **red** ▲ for the backward path and $k_B$. Additionally, matchings between forward and backward are displayed as **cyan** edges.

## 4.1 Pseudo-preimage Attack on SATURNIN-Hash

SATURNIN is one of the second-round candidates of the NIST LWC competition [CDL+20]. The block cipher has both a block and a key size of 256 bits, and it can be abstracted as an AES-like design operating on a 4×4 matrix of 16-bit nibbles. A single round of this representation is actually a "Super-round" that corresponds to two rounds in SATURNIN. It adds a key which alternates between the master key and a rotated version of it, then applies either a transformation on the rows (MixRows) or the columns (MixColumns). This can be viewed equivalently as the composition of a transposition, followed by a MixColumns, but the key bytes have to be positioned appropriately.

The hash function SATURNIN-Hash uses the block cipher as a compression function. In [DHS+21], the authors gave a pseudo-preimage attack on 7 super-rounds (out of 16) of time $2^{208}$ and memory $2^{48}$. We improve this time to $2^{192}$, at the expense of a larger memory, using the path of Figure 5.

We start by fixing the 4 key nibbles $k_{1,2,3,11}$. In the forward list, we put 6 nibbles $k_{5,6,7,8,9,10}$ and in the backward list, we put the 6 others $k_{0,4,12,13,14,15}$. We also fix the values of 14 state nibbles: 9 arrows, 2 linear relations through $c_0^3$, 1 linear relation through $c_0^5, c_1^5, c_2^5$ respectively.

The forward list is computed as follows: we start from $c_2^2$, which needs 4 nibble guesses. Going through $c_0^3$, the two nibbles are deduced using the two precomputed linear relations. The same is done through round 5, so there is nothing more to guess. The list is of size $2^{6\times16+4\times16} = 2^{160}$. The backward list guesses 3 nibbles at round 5 (the other ones are deduced by the linear relations), then one nibble at round 3, so it has the same size.

There are 10 linear relations between backward and forward: 4 relations through MixColumns at round 1, 2 relations which hadn't been used yet in $c_0^3$, and 4 matchings of nibbles. As we obtain a list of $2^{160}$ partial (pseudo)-preimages on 4 nibbles (the blue nibbles going through round 7) there remains to repeat this operation $2^{2\times16}$ times.

In the quantum setting, our tool finds an attack on the same number of rounds, of complexity $2^{115.55}$ and using $2^{32}$ QRAQM. The path is quite different, and corresponds to a classical attack with reduced memory. For completeness we include it in Appendix C.2.

## 4.2 Key-recovery Attack on full FUTURE

FUTURE [GPS22] is an AES-like cipher with 64-bit blocks and 128-bit keys. Its internal state is represented as a $4 \times 4$ matrix of 4-bit nibbles. The round function applies the following operations: • SubCells (S-Boxes); • MixColumns; • ShiftRows; • AddRoundKey. One can note that the order of operations is different from the AES; also, ShiftRows rotates the rows to the right (and not to the left like in the AES). There are 10 rounds; MixColumns is omitted at the last round.

The key addition alternates between two 64-bit subkeys $k^0, k^1$, where $k^0$ is first added in the "pre-round", and then, the key added at round $i$ is rotated by $5(i/2)$ bits.

Because the key bits are not aligned with the state nibbles, due to the rotations, we model the structure of FUTURE at the bit level. The composition of SubCells and MixColumns is viewed as a Super S-Box operating on 16 bits. When matching through MixColumns, groups of key bits that belong to the same S-Box should not have incompatible

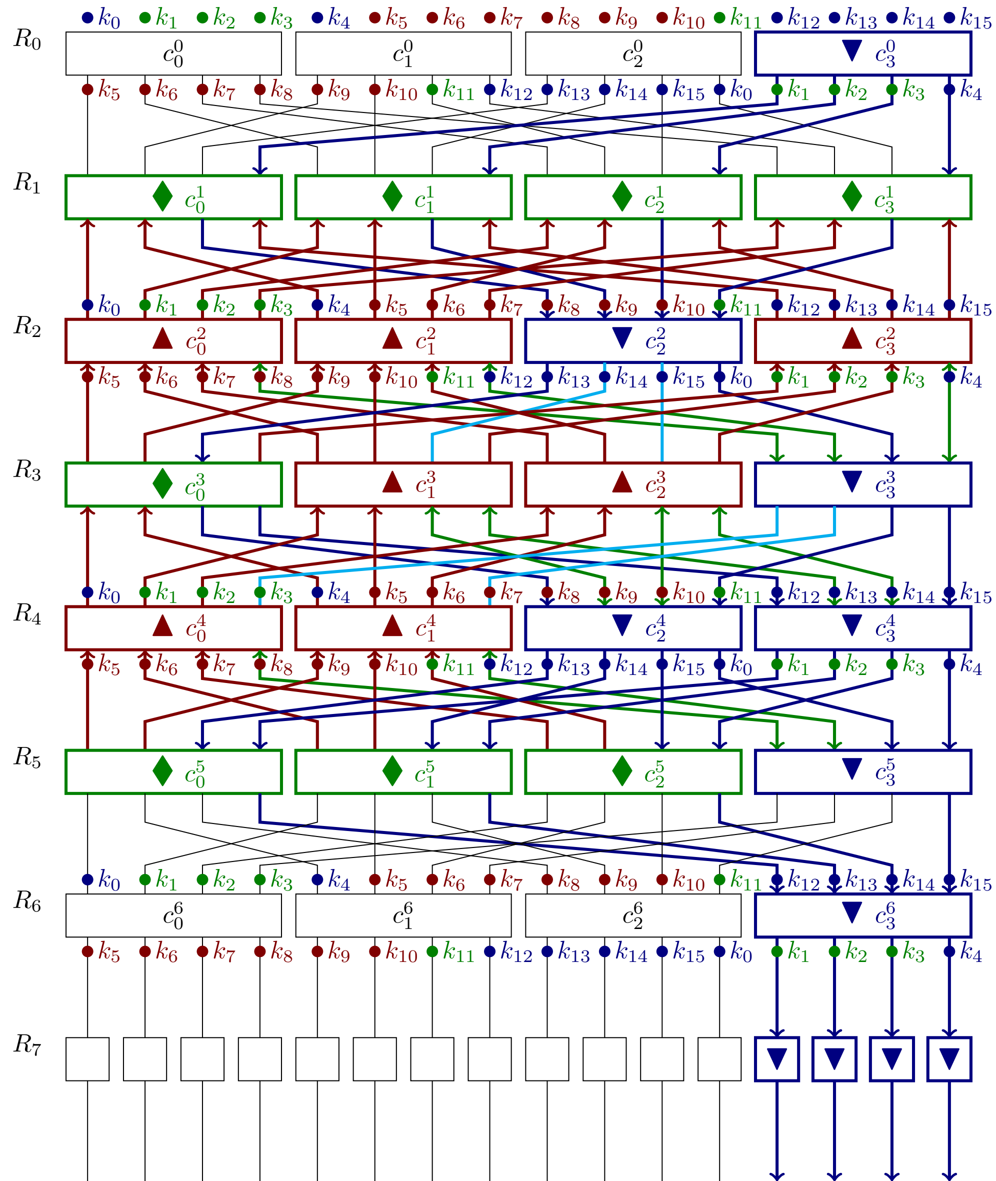


**Figure 5:** Path of a 7 Super-round pseudo-preimage attack on SATURNIN-Hash.

colors (i.e., both backward and forward). But since they come from the same cell, this is already ensured by the coloring constraints.

With our tool, we find an attack[4] of time complexity $2^{126}$ and memory $2^{34}$. The path is displayed in Figure 6. Due to space constraints, we have only represented the added key bits as color rectangles. Groups of 4 consecutive bits go through a single S-Box before the MixColumns operation.

In the forward path, we guess 14 key bits. In the backward path, we guess 22 key bits. The other 92 key bits are shared. There are 15 state nibbles guessed in total, and 1 nibble (4 bits) of linear relation at $c_3^2$.

Going forwards, we compute the states $c^1$ and then $c^2$ without having to make any state guess. At $c_3^2$ we use the single nibble of linear relation that we have precomputed; we have to guess 3 other nibbles. To compute $c^5$ we have to guess 2 nibbles. Thus the forward list is of total size: $2^{14+5\times 4} = 2^{34}$.

Going backwards, we have immediately the value of $c_1^1$ (the single red state nibble remaining is deduced by the precomputed linear relation, which relates it to the backward key bits). There is no other guess to make until round 7, where we need to guess 3 nibbles. Thus the list size is also $2^{34}$.

In the merged list, we can note that there are 3 nibbles of matching at round 2, since we only precomputed one linear relation. There are also 3 nibbles at round 6, going through MixColumns. Finally, there are 3 nibbles of matching at round 8. This makes a total of 36 bits of matching, so the merged list is of size $2^{32}$.

In [GPS22] the authors estimated that MITM attacks could reach up to 7 rounds of the cipher. However this estimate did not take into account the GAD technique. Indeed, in this attack, state guesses allow roughly to extend the forward path by two rounds, and the backward path by one round.

# 5 Application to Present-like Designs

In this section, we show some applications of our results to Present-like designs, where there is no Super S-Box. Because the designs attacked here involve either a large number of rounds or state nibbles, we only sketch briefly the structure of the attacks and give zoomed-out versions of some of the pictures. The full pictures can be automatically generated using our code.

## 5.1 Application to Present

Present [BKL+07] is an SPN cipher with a state of $16 \times 4$ bits. It uses a 4-bit S-Box, and the linear layer is a permutation of the bits. While the best key-recovery attacks on the block cipher Present are linear attacks, they have a large data complexity. The main advantage of MITM attacks is a small data complexity.

The Sieve-in-the-middle technique introduced in [CNV13] allowed to find an attack on 7-round Present-80 in time $2^{73.42}$. Using bicliques, this attack can be extended to 8 rounds with the same time complexity, and $2^6$ data complexity.

With our tool, we are able to obtain a GAD attack on 9 rounds with a comparable $2^8$ data complexity, and a time complexity $2^{78}$. A bird's eye view of the attack path is represented in Figure 7. The data complexity appears clearly since all state bits except 8 are fixed between the first round and the encryption cell.

We detail the choice of backward and forward key bits. Since each round of key scheduling applies an S-Box, there are a total of 9 key cells relating the key bits, and a total of $80 + 9 \times 4 = 116$ key bits. In the forward key, we take $k_{20}$, $k_{21}$, $k_{24}$, $k_{25}$ from the

[4]Actually, our optimization yields an even better attack with time complexity $2^{124}$ and memory $2^{84}$, but the path is much more complicated.

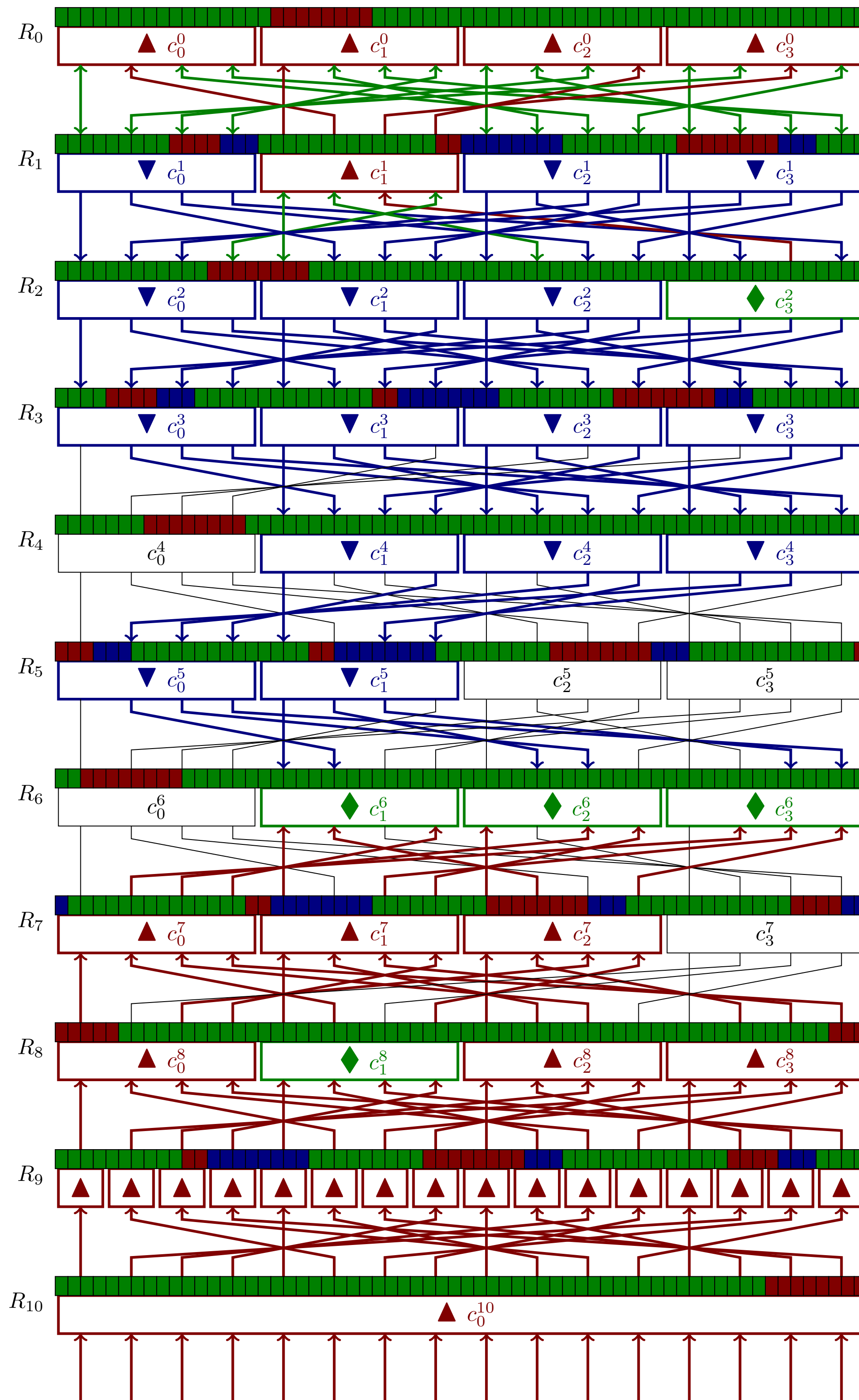


**Figure 6:** Path of the key-recovery attack on full FUTURE. Individual edges correspond to S-Boxes, which are aligned with groups of 4 key bits.

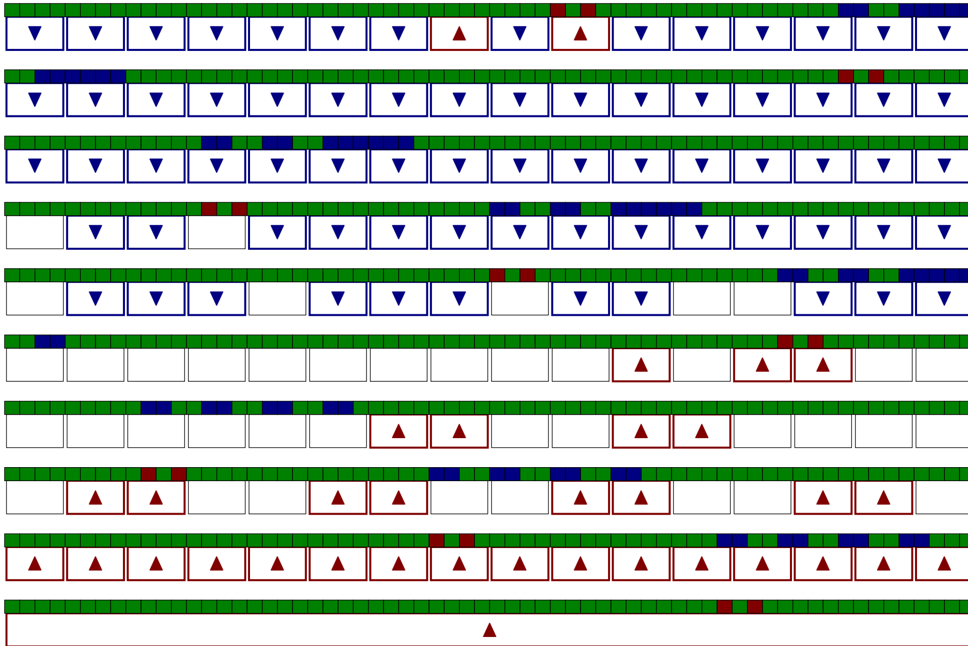

**Figure 7:** Low-data MITM attack on 9 rounds of PRESENT-80.

master key, and two key bits $k_{81}$, $k_{82}$ that appear on the left at the second round. Four key bits $k_{15}$, $k_{16}$, $k_{17}$, $k_{18}$, which are the remaining forward key bits visible at the first round, are deduced with a key cell relation.

All other key cells reduce the shared key bits. In the backward list, we only guess two key bits $k_{41}$, $k_{42}$ and there are no active key cells. Four bits of state guesses are required to activate the last two backward cells. All lists have size $2^6$. We can also fine-tune the trade-off between data and time, for example by taking a limit of $2^{12}$ data, we find a slightly smaller complexity $2^{77}$.

## 5.2 Application to FLY and PIPO

The lightweight block ciphers FLY [KG16] and PIPO [KJK+20] are structurally similar with respect to MITM attacks. Both are PRESENT-like SPN ciphers with 64-bit blocks, but 8-bit S-Boxes, where the linear layer is a bit-permutation. Though the S-Box and permutation differ between both designs, the cell-based representation will be the same. Indeed, what only matters for us is that between any pair of S-Boxes of two successive rounds, there is a single-bit linear relation.

This simplification is allowed by the simple key schedule of both ciphers. FLY uses a 128-bit master key which is equivalent to $k^0||k^1$, where $k^0$ is used in even rounds and $k^1$ in odd rounds. PIPO-128 does the same, while PIPO-256 has a 256-bit key which is separated in 4 subkeys $k = k^3||k^2||k^1||k^0$ used alternatively. The only difference between FLY and PIPO-128 is the final key addition in PIPO-128. Furthermore, in the attacks, the final key is always entirely known in the backward path. Applying a final bit permutation or not will lead to the same attack paths and complexities.

To the best of our knowledge, our attacks are the best against FLY and PIPO, exceeding the differential and linear cryptanalysis performed by the authors of PIPO (who reported 9-round and 11-round attacks respectively). We defer the details of the 128-bit versions to Appendix C.6 and focus here on our attack on full PIPO-256 (18 rounds).

Our attack has a time complexity $2^{252}$ and a memory complexity $2^{60}$. It uses the path

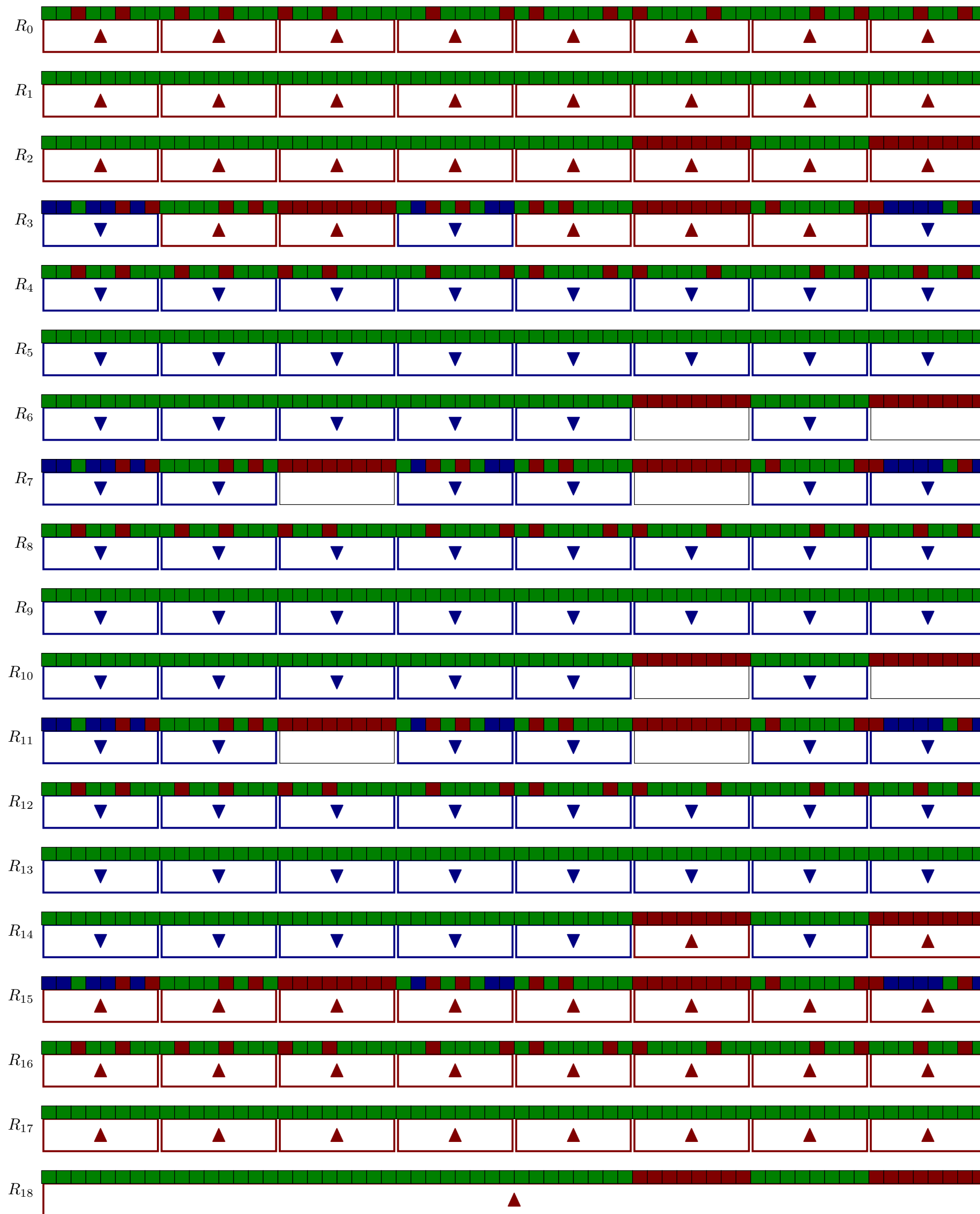


**Figure 8:** Path of the 18-round attack on PIPO-256.

given in Figure 8. There are 183 shared key bits, 13 forward and 60 backward. There are no state guesses backwards, as can be seen on the path. Forwards, we need to guess a total of 56 bits: 12 bits at round 7, 16 bits at round 8, 12 bits at round 11, 16 bits at round 12. Thus the list is of size $2^{69}$. In the middle, there are 64 bits of matching (a full state), so the merged list size is $2^{65}$. The overall complexity is dominated by the computation of the forward path: $2^{69+183} = 2^{252}$.

# 6 Conclusion

In this paper, we extended the cell-based modeling of [SS22a] to find MITM attacks on ciphers and compression functions with simple key schedules. The attacks obtained show that the MITM technique is quite powerful when the key schedule is weak, even if the cipher itself has a good diffusion.

At the moment, we conjecture that our modeling is optimal when the key-schedule has only S-Boxes and permutations as defined in Section 2. A natural open question would be to extend this further to the case of key schedules with complex linear layers, like AES. However, this seems to induce complex linear relations between the key nibbles and complex relations between state and key, which only a more fine-grained modeling can capture [BGST22]. For example, in preimage attacks on AES variants, one would like to guess nibbles or enforce constraints on linearly modified round keys (to have no impacts through MixColumns for example). Our model cannot find such constraints.

Regarding quantum MITM attacks, we have encountered instances of the Grover-meet-Simon attack [LM17] which cannot be turned “offline” [BHN+19]. Doing so would provide a valuable improvement of these attacks.

# Acknowledgments

We thank the reviewers of ToSC for their helpful feedback and comments. A.S. would like to thank Susanta Samanta for providing details on the specification of the block cipher FUTURE. Part of this work done at CWI by A.S. has been supported by ERC-ADG-ALGSTRONGCRYPTO (project 740972). This work has been partially supported by the French Agence Nationale de la Recherche through the DeCrypt project under Contract ANR-18-CE39-0007, and through the France 2030 program under grant agreement No. ANR-22-PETQ-0008 PQ-TLS.

# References

[AA20] Siavash Ahmadi and Mohammad Reza Aref. Generalized meet in the middle cryptanalysis of block ciphers with an automated search algorithm. *IEEE Access*, 8:2284–2301, 2020.

[AAMA14] Siavash Ahmadi, Zahra Ahmadian, Javad Mohajeri, and Mohammad Reza Aref. Low-data complexity biclique cryptanalysis of block ciphers with application to piccolo and HIGHT. *IEEE Trans. Inf. Forensics Secur.*, 9(10):1641–1652, 2014.

[AS08] Kazumaro Aoki and Yu Sasaki. Preimage attacks on one-block MD4, 63-step MD5 and more. In *Selected Areas in Cryptography*, volume 5381 of *LNCS*, pages 103–119. Springer, 2008.

[BDF11] Charles Bouillaguet, Patrick Derbez, and Pierre-Alain Fouque. Automatic search of attacks on round-reduced AES and applications. In *CRYPTO*, volume 6841 of *LNCS*, pages 169–187. Springer, 2011.

[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 *EUROCRYPT (1)*, volume 12696 of *LNCS*, pages 771–804. Springer, 2021.

[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 *CRYPTO (1)*, volume 13507 of *Lecture Notes in Computer Science*, pages 64–93. Springer, 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 *ASIACRYPT (1)*, volume 11921 of *Lecture Notes in Computer Science*, pages 552–583. Springer, 2019.

[BJ22] Xavier Bonnetain and Samuel Jaques. Quantum period finding against symmetric primitives in practice. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(1):1–27, 2022.

[BJK+16] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY family of block ciphers and its low-latency variant MANTIS. In *CRYPTO (2)*, volume 9815 of *Lecture Notes in Computer Science*, pages 123–153. Springer, 2016.

[BKL+07] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: an ultra-lightweight block cipher. In *CHES*, volume 4727 of *LNCS*, pages 450–466. Springer, 2007.

[BKR11] Andrey Bogdanov, Dmitry Khovratovich, and Christian Rechberger. Biclique cryptanalysis of the full AES. In *ASIACRYPT*, volume 7073 of *Lecture Notes in Computer Science*, pages 344–371. Springer, 2011.

[BNS19] Xavier Bonnetain, María Naya-Plasencia, and André Schrottenloher. Quantum security analysis of AES. *IACR Trans. Symmetric Cryptol.*, 2019(2):55–93, 2019.

[Bon21] Xavier Bonnetain. Tight bounds for simon's algorithm. In *LATINCRYPT*, volume 12912 of *Lecture Notes in Computer Science*, pages 3–23. Springer, 2021.

[BPP+17] Subhadeep Banik, Sumit Kumar Pandey, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, and Yosuke Todo. GIFT: A small present - towards reaching the limit of lightweight encryption. In *CHES*, volume 10529 of *Lecture Notes in Computer Science*, pages 321–345. Springer, 2017.

[BR10] Andrey Bogdanov and Christian Rechberger. A 3-subset meet-in-the-middle attack: Cryptanalysis of the lightweight block cipher KTANTAN. In *Selected Areas in Cryptography*, volume 6544 of *LNCS*, pages 229–240. Springer, 2010.

[CDL+20] Anne Canteaut, Sébastien Duval, Gaëtan Leurent, María Naya-Plasencia, Léo Perrin, Thomas Pornin, and André Schrottenloher. Saturnin: a suite of lightweight symmetric algorithms for post-quantum security. *IACR Trans. Symmetric Cryptol.*, 2020(S1):160–207, 2020.

[CLS22] Federico Canale, Gregor Leander, and Lukas Stennes. Simon's algorithm and symmetric crypto: Generalizations and automatized applications. In *CRYPTO (3)*, volume 13509 of *Lecture Notes in Computer Science*, pages 779–808. Springer, 2022.

[CNV13] Anne Canteaut, María Naya-Plasencia, and Bastien Vayssière. Sieve-in-the-middle: Improved MITM attacks. In *CRYPTO (1)*, volume 8042 of *LNCS*, pages 222–240. Springer, 2013.

[DF16] Patrick Derbez and Pierre-Alain Fouque. Automatic search of meet-in-the-middle and impossible differential attacks. In *CRYPTO (2)*, volume 9815 of *Lecture Notes in Computer Science*, pages 157–184. Springer, 2016.

[DH77] Whitfield Diffie and Martin E. Hellman. Special feature exhaustive cryptanalysis of the NBS data encryption standard. *Computer*, 10(6):74–84, 1977.

[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 *CRYPTO (3)*, volume 12827 of *LNCS*, pages 278–308. Springer, 2021.

[DR99] Joan Daemen and Vincent Rijmen. AES Proposal: Rijndael. Submission to the NIST AES competition, 1999.

[DR20] Joan Daemen and Vincent Rijmen. *The Design of Rijndael - The Advanced Encryption Standard (AES), Second Edition.* Information Security and Cryptography. Springer, 2020.

[DS08] Hüseyin Demirci and Ali Aydin Selçuk. A meet-in-the-middle attack on 8-round AES. In *FSE*, volume 5086 of *Lecture Notes in Computer Science*, pages 116–126. Springer, 2008.

[DSP07] Orr Dunkelman, Gautham Sekar, and Bart Preneel. Improved meet-in-the-middle attacks on reduced-round DES. In *INDOCRYPT*, volume 4859 of *LNCS*, pages 86–100. Springer, 2007.

[FKL+00] Niels Ferguson, John Kelsey, Stefan Lucks, Bruce Schneier, Michael Stay, David A. Wagner, and Doug Whiting. Improved cryptanalysis of rijndael. In *FSE*, volume 1978 of *Lecture Notes in Computer Science*, pages 213–230. Springer, 2000.

[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 *ASIACRYPT*, volume 6477 of *LNCS*, pages 56–75. Springer, 2010.

[GPS22] Kishan Chand Gupta, Sumit Kumar Pandey, and Susanta Samanta. FUTURE: A lightweight block cipher using an optimal diffusion matrix. In *AFRICACRYPT*, Lecture Notes in Computer Science, pages 28–52. Springer Nature Switzerland, 2022.

[Gro96] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *STOC*, pages 212–219. ACM, 1996.

[Gur23] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023.

[HE22] Hosein Hadipour and Maria Eichlseder. Autoguess: A tool for finding guess-and-determine attacks and key bridges. In *ACNS*, volume 13269 of *Lecture Notes in Computer Science*, pages 230–250. Springer, 2022.

[IS12] Takanori Isobe and Kyoji Shibutani. All subkeys recovery attack on block ciphers: Extending meet-in-the-middle approach. In *Selected Areas in Cryptography*, volume 7707 of *LNCS*, pages 202–221. Springer, 2012.

[KG16] Pierre Karpman and Benjamin Grégoire. The littlun s-box and the fly block cipher. In *NIST Lightweight Cryptography Workshop (informal proceedings)*, 2016.

[KJK+20] Hangi Kim, Yongjin Jeon, Giyoon Kim, Jongsung Kim, Bo-Yeon Sim, Dong-Guk Han, Hwajeong Seo, Seonggyeom Kim, Seokhie Hong, Jaechul Sung, and Deukjo Hong. PIPO: A lightweight block cipher with efficient higher-order masking software implementations. In *ICISC*, volume 12593 of *Lecture Notes in Computer Science*, pages 99–122. Springer, 2020.

[KLLN16a] Marc Kaplan, Gaëtan Leurent, Anthony Leverrier, and María Naya-Plasencia. Breaking symmetric cryptosystems using quantum period finding. In *CRYPTO (2)*, volume 9815 of *Lecture Notes in Computer Science*, pages 207–237. Springer, 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 v2 - efficient short-input hashing for post-quantum applications. *IACR Trans. Symmetric Cryptol.*, 2016(2):1–29, 2016.

[KRS12] Dmitry Khovratovich, Christian Rechberger, and Alexandra Savelieva. Bicliques for preimages: Attacks on skein-512 and the SHA-2 family. In *FSE*, volume 7549 of *LNCS*, pages 244–263. Springer, 2012.

[LJ16] Rongjia Li and Chenhui Jin. Meet-in-the-middle attacks on 10-round AES-256. *Des. Codes Cryptogr.*, 80(3):459–471, 2016.

[LM17] Gregor Leander and Alexander May. Grover meets simon - quantumly attacking the fx-construction. In *ASIACRYPT (2)*, volume 10625 of *Lecture Notes in Computer Science*, pages 161–178. Springer, 2017.

[MS22] Alexander May and Lars Schlieper. Quantum period finding is compression robust. *IACR Trans. Symmetric Cryptol.*, 2022(1):183–211, 2022.

[NC02] Michael A Nielsen and Isaac Chuang. Quantum computation and quantum information, 2002.

[QHD+23] Lingyue Qin, Jialiang Hua, Xiaoyang Dong, Hailun Yan, and Xiaoyun Wang. Meet-in-the-middle preimage attacks on sponge-based hashing. In *EUROCRYPT (4)*, volume 14007 of *Lecture Notes in Computer Science*, pages 158–188. Springer, 2023.

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

[Sas18] Yu Sasaki. Integer linear programming for three-subset meet-in-the-middle attacks: Application to GIFT. In *IWSEC*, volume 11049 of *LNCS*, pages 227–243. Springer, 2018.

[Sch23] André Schrottenloher. Quantum linear key-recovery attacks using the QFT. 14085:258–291, 2023.

[Sim97] Daniel R. Simon. On the power of quantum computation. *SIAM J. Comput.*, 26(5):1474–1483, 1997.

[SS17] Thomas Santoli and Christian Schaffner. Using simon's algorithm to attack symmetric-key cryptographic primitives. *Quantum Inf. Comput.*, 17(1&2):65–78, 2017.

[SS22a] André Schrottenloher and Marc Stevens. Simplified MITM modeling for permutations: New (quantum) attacks. In *CRYPTO (3)*, volume 13509 of *Lecture Notes in Computer Science*, pages 717–747. Springer, 2022.

[SS22b] André Schrottenloher and Marc Stevens. Simplified MITM modeling for permutations: New (quantum) attacks. *IACR Cryptol. ePrint Arch.*, page 189, 2022.

[SWWW12] Yu Sasaki, Lei Wang, Shuang Wu, and Wenling Wu. Investigating fundamental security requirements on whirlpool: Improved preimage and collision attacks. In *ASIACRYPT*, volume 7658 of *Lecture Notes in Computer Science*, pages 562–579. Springer, 2012.

# A Complete MILP Model with Keys

We give the generic MILP model for the GAD case, possibly with some matching through MixColumns. Matching through MC is possible only at the cells which are indicated to be Super S-Boxes. The pseudocode that we give below is a slightly different and simplified version of the file `modeling.py` available at https://github.com/AndreSchrottenloher/key-mitm.

The input is given as a graph. We have a set of cells $C$ with weights $w_c$, a set of key nibbles $K$, a set of edges $E \subseteq C \times C \times \mathbb{N} \times K$ which have both a weight and a key nibble attached, a set of key cells $KC$ also with weights, and corresponding input and output key nibbles. We assume that all edges and key nibbles have weight 1.

Since this modeling is built over [SS22a], we indicate the new parts that are related to key nibbles in orange (lines 10–31). The variable names $|x|, |k_F|, |k_S|, |k_B|, \ell_F, \ell_B, \ell_M$ are taken in order to connect immediately to the formulas of Equation 8 and Equation 9.

▷ Variables for the coloration of cells and basic constraints on the path
1: **for all** cell $c \in C$ **do**
2: **Variable** (boolean) $\mathsf{cell_col}_L[c]$ for $L \in \{F, B, M, N\}$
▷ Forward, backward, merged cells and also "new merged" cells which can appear if the cell is a Super S-Box
3: **Constraint** $\mathsf{cell_col}_M[c] = \mathsf{cell_col}_F[c] + \mathsf{cell_col}_B[c] + \mathsf{cell_col}_N[c]$
4: If $c$ is not a Super S-Box, **Constraint** $\mathsf{cell_col}_N[c] = 0$
5: **end for**
6: For all round $r$, **Constraint** "no $N$ cell for both $r$ and $r+1$"
7: **Constraint** "at least one round without $B$ cell"
8: **Constraint** "at least one round without $F$ cell"
9: **Constraint** "at least one round with all cells $M$"

▷ Variable and constraints on the coloration of key nibbles

10: **for all** key nibble $k \in K$ **do**
11: **Variable** (boolean) $\mathsf{key_col}_L[k]$ for $L \in \{F, B, S\}$
12: **Constraint** $\mathsf{key_col}_B[k] + \mathsf{key_col}_F[k] + \mathsf{key_col}_S[k] = 1$
13: **end for**
14: **for all** edge $(c, c', w, k)$ **do**
▷ If both cells are forward (or "new merged" for the below cell) then key nibble is forward or shared
15: **Constraint** $1+\mathsf{key_col}_F[k]+\mathsf{key_col}_S[k] \geq \mathsf{cell_col}_F[c]+\mathsf{cell_col}_N[c']+\mathsf{cell_col}_F[c']$
▷ If both cells are backward (or "new merged" for the below cell) then key nibble is backward or shared
16: **Constraint** $1+\mathsf{key_col}_B[k]+\mathsf{key_col}_S[k] \geq \mathsf{cell_col}_B[c]+\mathsf{cell_col}_N[c']+\mathsf{cell_col}_B[c']$
17: **end for**
▷ Constraints on the coloration of key cells
18: **for all** key cell $kc$ **do**
19: **Variable** (boolean) $\mathsf{key_cell_col}_L[kc]$ for $L \in \{F, B, S\}$
20: **Constraint** $\mathsf{key_cell_col}_S[kc] + \mathsf{key_cell_col}_F[kc] + \mathsf{key_cell_col}_B[kc] \leq 1$
21: **for** $k$ in input and output key nibbles of $kc$ **do**
22: **Constraint** $\mathsf{key_col}_S[k] \geq \mathsf{key_cell_col}_S[kc]$
23: **Constraint** $\mathsf{key_col}_S[k] + \mathsf{key_col}_F[k] \geq \mathsf{key_cell_col}_F[kc]$
24: **Constraint** $\mathsf{key_col}_S[k] + \mathsf{key_col}_B[k] \geq \mathsf{key_cell_col}_B[kc]$
25: **end for**
26: **end for**
▷ Total amount of key nibbles
27: **Variable** (integer) $|k_L|$ for $L \in \{F, B, M, S\}$
28: **for** $L \in \{F, B, S\}$ **do**
29: **Constraint** $|k_L| = \sum_{k \in K} \mathsf{key_col}_L[k] - \sum_{kc \in KC} w(kc)\mathsf{key_cell_col}_L[kc]$
30: **end for**
31: **Constraint** $|k_M| = |k_F| + |k_B|$
▷ "Global reduction" variables
32: **for all** cell $c \in C$ **do**
33: **Variable** $\mathsf{global_red}[c]$ (continuous, between 0 and $w_c$)
▷ Global reductions only happen at forward (guessed edges) and "new merged" (through MC) cells. The following limit holds in both cases:
34: $\mathsf{global_red}[c] \leq$ (connections with backward cells at round before)
▷ "Connections" are computed by weighted sums of cell coloration variables
35: **if** $c$ is a super S-Box **then**
36: $\mathsf{global_red}[c] \leq w_c(\mathsf{cell_col}_N[c] + \mathsf{cell_col}_F[c])$
▷ For a "new merged" cell, it depends on the number of connections to other backward and forward cells
37: $\mathsf{global_red}[c] \leq$ (connections with colored cells) $- w_c \mathsf{cell_col}_N[c]$
38: $\mathsf{global_red}[c] \leq w_c \mathsf{cell_col}_F[c] +$ (connections with forward cells at round after)
39: **else**
40: $\mathsf{global_red}[c] \leq w_c \mathsf{cell_col}_F[c]$
41: **end if**
42: **end for**
43: $|x| = \sum_{c \in C} \mathsf{global_red}[c]$
▷ List sizes
44: **for** $c \in C$ **do**
45: **Variable** (continuous) $\mathsf{cell_contrib}_L[c]$ for $L \in \{F, B, M\}$
46: $\mathsf{cell_contrib}_L[c] = w_c \mathsf{cell_col}_L[c] - \left( \begin{array}{c} \text{connections with cells of} \\ \text{the same color at round before} \end{array} \right)$
47: **end for**

48: $\ell_L = (\sum_c \mathsf{cell_contrib}_L[c]) - |x|$
49: Memory complexity: $\min(\ell_F + |k_F|, \ell_B + |k_B|)$
50: Time complexity: determined from $\ell_F, \ell_B, \ell_M, |x|, |k_B|, |k_F|, |k_S|$ using the formulas of Equation 8 and Equation 9.

# B Attack Algorithms

In this section, we illustrate the conversion of a MITM attack path (found with our model) into a MITM attack algorithm on the two examples of Section 4. The attacks of Section 5 are easier to write down since they do not involve matching through MixColumns.

## B.1 Pseudo-preimage Attack on Saturnin-Hash

We detail the algorithm of the attack given in Section 4.1 (Figure 5). Recall that a single super-round of Saturnin applies the following operations: AddRoundKey, SubBytes, MixColumns (resp. MixRows). The attack does not depend on the definition of the (super) S-Box, nor the mixing layer (as long as it uses an MDS matrix). Below we simply need to write linear equations that relate the input and output nibbles of the mixing operation; we do not detail these equations and simply introduce linear functions of the nibbles. We use $S$ to denote the super S-Box.

We use the following notations to coincide with the figure: $x_i^j$ for the column $i$ after the key addition at round $j$ (the arrows entering cell $c_i^j$) and $z_i^j$ after the mixing layer (the arrows exiting cell $c_i^j$). We also reuse the forward / backward / {merged, shared} coloring scheme in the writing to make the separation of the forward and backward computations clearer.

**Path Details.** The forward, backward and shared key nibbles are:

$$\begin{cases} k_F = k_0, k_4, k_{13}, k_{13}, k_{14}, k_{15} \\ k_B = k_6, k_6, k_7, k_8, k_9, k_{10} \\ k_S = k_1, k_2, k_3, k_{11} \end{cases} . \quad (10)$$

We fix a total of 14 state nibbles, including 9 arrows and 5 linear relations through the mixing layer. We write equations depending on linear functions $L_i, L'_i$, whose exact specification depends on the mixing layer. For example, for the matching through $c_0^1$, we relate outputs of the cells at round 0 ($z$ variables) with inputs of the cells at round 2 ($x$ variables):

$$L(S(k_1 \oplus z_3^0[0]), x_2^2[0] \oplus k_8) = L'(x_0^2[0] \oplus k_0, x_1^2[0] \oplus k_4, x_2^2[0] \oplus k_{12})$$
$$\implies L_1(S(k_1 \oplus z_3^0[0]), x_2^2[0], k_0, k_4, k_{12}) = L'_1(k_8, x_0^2[0], x_1^2[0], x_2^2[0]) \quad (11)$$

Similarly through $c_1^1$, $c_2^1$, $c_3^1$ respectively:

$$L_2(S(k_2 \oplus z_3^0[1]), x_2^2[1], k_{13}) = L'_2(k_1 \oplus x_0^2[1], k_5 \oplus x_1^2[1], k_9, x_3^2[1])$$
$$L_3(S(k_3 \oplus z_3^0[2]), x_2^2[2], k_{14}) = L'_3(k_2 \oplus x_0^2[2], k_6 \oplus x_1^2[2], x_3^2[2], k_{10})$$
$$L_4(S(k_4 \oplus z_3^0[3]), k_{11} \oplus x_2^2[3], k_{15}) = L'_4(k_3 \oplus x_0^2[3], k_7 \oplus x_1^2[3], x_2^2[3])$$

Through $c_0^3$ we have two linear relations to precompute, for which we select appropriate choices of input and output nibbles:

$$L(S(x_0^3[0]), S(x_0^3[2]), z_0^3[0], z_0^3[2], z_0^3[3]) = 0$$
$$\implies L(S(k_5 \oplus z_0^2[0]), S(k_{13} \oplus z_2^2[0]), k_0 \oplus x_0^4[0], k_8 \oplus x_2^4[0], k_{12} \oplus x_3^4[0]) = 0$$
$$\implies L_5(S(k_{13} \oplus z_2^2[0]), k_0, x_2^4[0], k_{12} \oplus x_3^4[0]) = L'_5(S(k_5 \oplus z_0^2[0]), x_0^4[0], k_8) := t_5 \quad (12)$$

$$L'(S(x_0^3[1]), S(x_0^3[2]), z_0^3[0], z_0^3[2], z_0^3[3]) = 0$$
$$\implies L_6(S(k_{13} \oplus z_2^2[0]), k_0, x_2^4[0], k_{12} \oplus x_3^4[0]) = L'_6(S(k_9 \oplus z_1^2[0]), x_0^4[0], k_8) := t_6 \quad (13)$$

and we can write two additional equations for matching, using the rest of the nibbles. Finally we write the linear relations precomputed through $c_0^5, c_1^5, c_2^5$:

$$L_7(S(k_{13} \oplus z_2^4[0]), S(k_1 \oplus z_3^4[0]), k_{12} \oplus x_3^6[0]) = L'_7(S(k_5 \oplus z_0^4[0]), S(k_9 \oplus z_1^4[0])) := t_7 \quad (14)$$

$$L_8(S(k_{14} \oplus z_2^4[1]), S(k_2 \oplus z_3^4[1]), k_{13} \oplus x_3^6[1]) = L'_8(S(k_6 \oplus z_0^4[1]), S(k_{10} \oplus z_1^4[1])) := t_8 \quad (15)$$

$$L_9(S(k_{15} \oplus z_2^4[2]), S(k_3 \oplus z_3^4[2]), k_{14} \oplus x_3^6[2]) = L'_9(S(k_7 \oplus z_0^4[2]), S(k_{11} \oplus z_1^4[2])) := t_9 \quad (16)$$

The additional 9 state nibble guesses (green arrows) are:

$$x_3^3[0], z_1^2[3], z_3^2[3], x_2^4[1,2], z_1^3[3], z_2^3[3], x_3^5[0], z_1^4[3] \quad (17)$$

Notice that we alternate between $z$ and $x$ nibbles depending on the color of the key nibble that is added. Indeed, if the key nibble added on the edge is forward, then the state nibble must be fixed *before* the key addition (otherwise we wouldn't be able to compute forwards). If it is backward, the state nibble must be fixed *after* the key addition.

**Forward and Backward Computations.** The forward path (Algorithm 1) starts at round 2 and stops at round 0. The backward path (Algorithm 2) starts at round 4 and stops at round 2.

## B.2 Key-recovery Attack on FUTURE

We detail the algorithm of the attack given in Section 4.2 (Figure 6). Recall that a single round of Future applies the following operations: SubCells, MixColumns, ShiftRows, AddRoundKey, and there is a "pre-round" key addition before round 0. The key addition alternates between $k^0$ and $k^1$ where $k^0$ is first added in the pre-round.

The attack does not depend on the specification of the S-Box nor the MixColumns, as soon as the matrix is MDS. Like above, we will introduce many linear functions $L_i, L'_i$ without giving their full specification. The S-Box is denoted $S$.

**Notation.** Our notation coincides with Figure 6 as follows. Rounds are numbered from 0 to 9. We use $(x_0^i, x_1^i, x_2^i, x_3^i)$ to denote the 4 columns of the state at the beginning of round $i$, thus $(x_0^0, x_1^0, x_2^0, x_3^0)$ is the state after the pre-round key addition, and before the first S-Box layer. We denote by $(z_0^i, z_1^i, z_2^i, z_3^i)$ the state after the MC layer of round $i$. On the figure this means that $x_j^i$ is the value on the top of cell $c_j^i$ (incoming edges), and $z_j^i$ the value on the bottom (outgoing edges).

For a column $x_j^i$, we use $x_j^i[u]$ $(0 \leq u < 4)$ to denote individual 4-bit nibbles in this column. The SR operation transforms the state as follows:

$$\begin{pmatrix} z_0[0] & z_1[0] & z_2[0] & z_1[0] \\ z_0[1] & z_1[1] & z_2[1] & z_3[1] \\ z_0[2] & z_1[2] & z_2[2] & z_3[2] \\ z_0[3] & z_1[3] & z_2[3] & z_3[3] \end{pmatrix} \mapsto \begin{pmatrix} z_0[0] & z_1[0] & z_2[0] & z_3[0] \\ z_3[1] & z_0[1] & z_1[1] & z_2[1] \\ z_2[2] & z_3[2] & z_0[2] & z_1[2] \\ z_1[3] & z_2[3] & z_3[3] & z_0[3] \end{pmatrix} \quad (18)$$

We use $x_j^i[u][v]$ $(0 \leq v < 4)$ to denote individual bits within a nibble. The key bits are denoted $k^j[i]$ for $j \in \{0,1\}$ and $i \in \{0,\ldots,63\}$. Finally, multiple bits (resp. nibbles) will

**Algorithm 1** Forward computation for the attack on Saturnin-Hash.

**Global:** guess of $t_5$ to $t_9$, 9 state nibbles
**Input:** $k_F$ (6 key nibble guesses) and 4 state guesses
**Output:** 10 nibbles for matching

1: **procedure** $f_F$
2: Guess $x_2^2$ (use 4 guesses)
3: Deduce $x_3^3$ (use 3 global state nibbles)
4: Use Equation 12 and Equation 13 to deduce both $x_3^4[0]$ and $x_2^4[0]$ from the incoming edge $z_2^2[1]$
5: Deduce $x_2^4, x_3^4$ (use 4 global state nibbles)
6: Deduce $x_3^5$ (use 2 global state nibbles)
7: Using the three linear relations through $c_0^5, c_1^5, c_2^5$ (Equation 14 to Equation 16) deduce $x_3^6$
8: XOR with the target preimage and advance to $x_3^0$
9: **Return** the following values:
Through $c_0^1$: $L_1(S(k_1 \oplus z_3^0[0]), x_2^2[0], k_0, k_4, k_{12})$
Through $c_1^1$: $L_2(S(k_2 \oplus z_3^0[1]), x_2^2[1], k_{13})$
Through $c_2^1$: $L_3(S(k_3 \oplus z_3^0[2]), x_2^2[2], k_{14})$
Through $c_3^1$: $L_4(S(k_4 \oplus z_3^0[3]), k_{11} \oplus x_2^2[3], k_{15})$
Direct match: $k_{14} \oplus z_2^2[1]$
Direct match: $k_{15} \oplus z_2^2[2]$
Direct match: $z_3^3[0]$
Direct match: $z_3^3[1]$
Through $c_0^3$: Two additional relations
10: **end procedure**

**Algorithm 2** Backward computation for the attack on Saturnin-Hash.

**Global:** guess of $t_5$ to $t_9$, 9 state nibbles
**Input:** $k_B$ (6 key nibble guesses) and 4 state guesses
**Output:** 10 nibbles for matching

1: **procedure** $f_B$
2: Guess $z_0^4[0,1,2]$ (use 3 guesses)
3: Using the three linear relations through $c_0^5, c_1^5, c_2^5$ (Equation 14 to Equation 16) deduce $z_1^4[0,1,2]$
4: Complete $z_0^4$ and $z_1^4$ (use 2 global state nibbles)
5: Compute $z_1^3$ and $z_2^3$ (use 4 global state nibbles)
6: Guess $x_0^4[0]$ (use 1 guess)
7: Using the two precomputed relations through $c_0^3$ (Equation 12 and Equation 13), deduce $z_0^2[0]$ and $z_1^2[0]$
8: Compute $z_0^2, z_1^2, z_3^2$ (use 3 global state nibbles)
9: **Return** the following values:
Through $c_0^1$: $L_1'(k_8, x_0^2[0], x_1^2[0], x_2^2[0])$
Through $c_1^1$: $L_2'(k_1 \oplus x_0^2[1], k_5 \oplus x_1^2[1], k_9, x_3^2[1])$
Through $c_2^1$: $L_3'(k_2 \oplus x_0^2[2], k_6 \oplus x_1^2[2], x_3^2[2], k_{10})$
Through $c_3^1$: $L_4'(k_3 \oplus x_0^2[3], k_7 \oplus x_1^2[3], x_2^2[3])$
Direct match: $x_1^3[2]$
Direct match: $x_2^3[2]$
Direct match: $k_3 \oplus x_0^4[3]$
Direct match: $k_7 \oplus x_1^4[3]$
Through $c_0^3$: Two additional relations
10: **end procedure**

be denoted either $x[u, u']$ or $x[u{-}u']$ for a range of values $\{u, \ldots, u'\}$. As an example, the state at the beginning of round 1 is related to the state at round 0 by:

$$\begin{pmatrix} x_0^1[0] & x_1^1[0] & x_2^1[0] & x_3^1[0] \\ x_0^1[1] & x_1^1[1] & x_2^1[1] & x_3^1[1] \\ x_0^1[2] & x_1^1[2] & x_2^1[2] & x_3^1[2] \\ x_0^1[3] & x_1^1[3] & x_2^1[3] & x_3^1[3] \end{pmatrix}$$
$$= \begin{pmatrix} z_0^0[0] \oplus k^1[0{-}3] & z_1^0[0] \oplus k^1[16{-}19] & z_2^0[0] \oplus k^1[32{-}35] & z_3^0[0] \oplus k^1[48{-}51] \\ z_3^0[1] \oplus k^1[4{-}7] & z_0^0[1] \oplus k^1[20{-}23] & z_1^0[1] \oplus k^1[36{-}39] & z_2^0[1] \oplus k^1[52{-}55] \\ z_2^0[2] \oplus k^1[8{-}11] & z_3^0[2] \oplus k^1[24{-}27] & z_0^0[2] \oplus k^1[40{-}43] & z_1^0[2] \oplus k^1[56{-}59] \\ z_1^0[3] \oplus k^1[12{-}15] & z_2^0[3] \oplus k^1[28{-}31] & z_3^0[3] \oplus k^1[44{-}47] & z_0^0[3] \oplus k^1[60{-}63] \end{pmatrix}$$

**Path Details.** In the attack, the forward and backward key bits are:

$$\begin{cases} k_F := k^1[13{-}15, 32{-}39, 57{-}59], |k_F| = 14 \\ k_B := k^0[17{-}24], k^1[9{-}12, 30, 31, 49{-}56], |k_B| = 22 \end{cases} \quad (19)$$

and the shared key bits $k_S$ are the 92 remaining others.

The 15 state nibbles which are fixed (cut set) are the following:

$$X := x_0^1[0,1,2], \underbrace{x_0^1[3][0], z_1^0[3][1,2,3]}, z_2^0[0], z_1^0[1], x_2^1[2,3], x_3^1[0,1,3],$$
$$\underbrace{x_3^1[2][0], z_1^0[2][1,2,3]}, x_0^2[3], x_1^2[0], x_2^2[1] \ . \quad (20)$$

In addition, we will precompute one linear relation through MixColumns. Here, notice that we cannot simply fix $z_1^0[3]$ and $z_1^0[2]$, because the key bits on this nibble have both forward and backward colors. The backward key bit addition needs to be placed *before* the cut set, and the forward key addition *after*.

**Equations for Matching.** We will match through MixColumns in the cells $c_3^2, c_1^6, c_2^6, c_3^6, c_1^8$. Therefore we write linear equations between the input and output nibbles in these cells, and we introduce a nibble variable $t$. Note that in all these equations, we needed to separate the backward and forward / shared bits of the key nibbles, and the linearity of the MixColumns operation is essential. We will write for example: $k^1[29, 30, 31, 32] = (k^1[29]|0|0|k^1[32]) \oplus (0|k^1[30, 31]|0)$ where the zeroes are zero bits and | denotes concatenation of bits.

At $c_3^2$:

$$S(x_3^2[2]) = L(z_3^2[0,1,2,3])$$
$$S(x_3^2[2]) = L(k^1[9{-}12] \oplus x_0^3[1], k^1[49{-}52] \oplus x_2^3[3], k^1[53{-}56] \oplus x_3^3[0], k^1[29,30,31,32] \oplus x_1^3[2])$$
$$\Longrightarrow L_1(x_0^3[1], x_2^3[3], x_3^3[0], (k^1[29]|0|0|k^1[32]) \oplus x_1^3[2])$$
$$= L_1'(S(x_3^2[2]), k^1[9{-}12], k^1[49{-}52], k^1[53{-}56], (0|k^1[30,31]|0)) := t \quad (21)$$

And similarly, with a different separation of the values:

$$L_2(S(x_3^2[0]), x_0^3[1], x_2^3[3], x_3^3[0], (k^1[29]|0|0|k^1[32]) \oplus x_1^3[2])$$
$$= L_2'(k^1[9{-}12], k^1[49{-}52], k^1[53{-}56], (0|k^1[30,31]|0)) \quad (22)$$

$$L_3(S(x_3^2[1]), x_0^3[1], x_2^3[3], x_3^3[0], (k^1[29]|0|0|k^1[32]) \oplus x_1^3[2])$$
$$= L_3'(k^1[9{-}12], k^1[49{-}52], k^1[53{-}56], (0|k^1[30,31]|0))$$
$$(23)$$

$$L_4(S(x_3^2[3]), x_0^3[1], x_2^3[3], x_3^3[0], (k^1[29]|0|0|k^1[32]) \oplus x_1^3[2]) \\ = L_4'(k^1[9{-}12], k^1[49{-}52], k^1[53{-}56], (0|k^1[30,31]|0)) \tag{24}$$

Next, at $c_1^6$:

$$\begin{aligned} & L_5(S(x_1^6[0]), S(x_1^6[1])) = L_5'(z_1^6[0,1,3]) \\ & \implies \begin{cases} L_5(S(x_1^6[0]), S(x_1^6[1])) \\ \quad = L_5'(k^1[31, 32{-}34] \oplus x_1^7[0], k^1[51{-}54] \oplus x_2^7[1], k^1[27{-}29, 30] \oplus x_0^7[3]) \end{cases} \\ & \implies \begin{cases} L_5(S(x_1^6[0]), S(x_1^6[1])) \oplus L_5'((0|k^1[32{-}34]), 0, 0) \\ \quad = L_5'((k^1[31]|0|0|0) \oplus x_1^7[0], k^1[51{-}54] \oplus x_2^7[1], k^1[27{-}29, 30] \oplus x_0^7[3]) \end{cases} \end{aligned} \tag{25}$$

At $c_2^6$:

$$\begin{cases} L_6(S(x_2^6[1]), S(x_2^6[2])) \\ \quad = L_6'(k^1[47, 48, 49, 50] \oplus x_2^7[0], k^1[23{-}26] \oplus x_0^7[2], k^1[43{-}46] \oplus x_1^7[3]) \end{cases} \tag{26}$$

At $c_3^6$:

$$\begin{aligned} & L_7(S(x_3^6[2]), S(x_3^6[3])) = L_7'(k^1[19{-}22] \oplus x_0^7[1], k^1[39, 40{-}42] \oplus x_1^7[2], k^1[59, 60{-}62] \oplus x_2^7[3]) \\ & \implies L_7(S(x_3^6[2]), S(x_3^6[3])) \oplus L_7'(0, k^1[39]|0|0|0, k^1[59]|0|0|0) \\ & = L_7'(k^1[19{-}22] \oplus x_0^7[1], (0|k^1[40{-}42]) \oplus x_1^7[2], (0|k^1[60{-}62]) \oplus x_2^7[3]) \end{aligned} \tag{27}$$

And finally at $c_1^8$:

$$\begin{aligned} & S(x_1^8[0]) = L_8(k^1[32{-}35] \oplus x_0^9[3], k^1[36{-}39] \oplus x_1^9[0], k^1[56, 57{-}59] \oplus x_2^9[1], \\ & \qquad k^1[12, 13{-}15] \oplus x_3^9[2]) \\ & \implies \begin{cases} L_8(k^1[32{-}35], k^1[36{-}39], 0|k^1[57{-}59], 0|k^1[13{-}15]) = \\ \quad S(x_1^8[0]) \oplus L_8(x_0^9[3], x_1^9[0], (k^1[56]|0|0|0) \oplus x_2^9[1], (k^1[12]|0|0|0) \oplus x_3^9[2]) \end{cases} \end{aligned} \tag{28}$$

and two similar equations where we replace $S(x_1^8[0])$ by $S(x_1^8[1])$ (resp. $S(x_1^8[2])$) and $L_8$ by another linear function $L_9$ (resp. $L_{10}$).

**Forward and Backward Computations.** The forward and backward computations are detailed in Algorithm 3 and Algorithm 4. There are 9 nibble conditions of matching, i.e., 36 bits (so the merged list is of expected size $2^{32}$). One can note that some of the matching conditions are independent from others; for example the three first equations on the backward path depend only on the key, and not on the state guesses.

# C Details on Applications

In this section, we give some details of applications which were omitted from Section 4 and Section 5.

## C.1 Application to 1k-AES and 2k-AES

We illustrate the power of GAD with the following structures: • 1k-AES: the AES block cipher with a 128-bit key $k$, without key schedule; • 2k-AES: the AES block cipher with alternating 128-bit keys $k^0, k^1$.

**Algorithm 3** Forward computation for the attack on FUTURE.

**Global:** $k_S, X, t$
**Input:** 14 bits of $k_F$, 20 bits for state guesses
**Output:** 9 nibbles for matching

1: **procedure** $f_F$
2: Using the value of $X$ and the value of $k_F$, compute:

$$\begin{cases} x_0^1 = x_0^1[0,1,2], x_0^1[3][0] | (k^1[13,14,15] \oplus z_1^0[3][1,2,3]) \\ x_2^1 = k^1[32{-}35] \oplus z_2^0[0], k^1[36{-}39] \oplus z_1^0[1], x_2^1[2,3] \\ x_3^1 = x_3^1[0,1,2], x_3^1[2][0] | (k^1[57{-}59] \oplus z_1^0[2][1,2,3]) \end{cases}$$

3: Deduce $x_0^2, x_1^2, x_2^2$ except the nibbles $x_0^2[3], x_1^2[0], x_2^2[1]$
4: Use $X$ to complete $x_0^2[3], x_1^2[0], x_2^2[1]$; guess $x_0^3[1], x_2^3[3], x_3^3[0]$ ▷ 12 bits of state guess
▷ These states are unknown because backward key nibbles were added to them
5: Deduce $x_1^3[2]$ by Equation 21: $L'_1(x_0^3[1], x_2^3[3], x_3^3[0], x_1^3[2] \oplus (k^1[29]|0|0|k^1[32])) = t$
6: Compute forwards to $x_1^4, x_2^4, x_3^4$ using $k_S$
7: Guess $x_0^5[0], x_1^5[1]$ ▷ 8 bits of state guess
8: Deduce $x_0^5, x_1^5$
9: **Return** the following values:
▷ They correspond to the left hand sides of the matching equations
Equation 22: $L_2(S(x_3^2[0]), x_0^3[1], x_2^3[3], x_3^3[0], (k^1[29]|0|0|k^1[32]) \oplus x_1^3[2])$
Equation 23: $L_3(S(x_3^2[1]), x_0^3[1], x_2^3[3], x_3^3[0], (k^1[29]|0|0|k^1[32]) \oplus x_1^3[2])$
Equation 24: $L_4(S(x_3^2[3]), x_0^3[1], x_2^3[3], x_3^3[0], (k^1[29]|0|0|k^1[32]) \oplus x_1^3[2])$
Equation 25: $L_5(S(x_1^6[0]), S(x_1^6[1])) \oplus L'_5((0|k^1[32{-}34]), 0, 0)$
Equation 26: $L_6(S(x_2^6[1]), S(x_2^6[2]))$
Equation 27: $L_7(S(x_3^6[2]), S(x_3^6[3])) \oplus L'_7(0, k^1[39]|0|0|0, k^1[59]|0|0|0)$
Equation 28: $L_8(k^1[32{-}35], k^1[36{-}39], 0|k^1[57{-}59], 0|k^1[13{-}15])$
Equation 28: $L_9(k^1[32{-}35], k^1[36{-}39], 0|k^1[57{-}59], 0|k^1[13{-}15])$
Equation 28: $L_{10}(k^1[32{-}35], k^1[36{-}39], 0|k^1[57{-}59], 0|k^1[13{-}15])$
10: **end procedure**

**Algorithm 4** Backward computation for the attack on FUTURE.

**Global:** $k_S, X, t$
**Input:** 22 bits for $k_B$, 12 bits for state guesses
**Output:** 9 nibbles for matching

1: **procedure** $f_B$
2: Using Equation 21, $t$ and $k_B$, deduce $x_3^2[2]$
3: Using the value of $X$ and the value of $k_B$, compute:

$$z_1^1[0{-}3] = k^0[21{-}24] \oplus x_1^2[0], k^0[41{-}44] \oplus x_2^2[1], k^0[61{-}63, 0] \oplus x_3^2[2], k^0[17{-}20] \oplus x_0^2[3]$$

4: Using the value of $k_B$ and $k_S$, deduce $z_0^0[1], z_1^0[0], z_2^0[3], z_3^0[2]$
5: Complete the state at round 0:

$$\begin{cases} z_0^0[0,2,3] = k^1[0{-}3] \oplus x_0^1[0], k^1[40{-}43] \oplus x_2^1[2], k^1[60{-}63] \oplus x_3^1[3] \\ z_1^0[1,2,3] = z_1^0[1], (k^1[56] \oplus x_3^1[2][0]) | z_1^0[2][1,2,3], (k^1[12] \oplus x_0^1[3][0]) | z_1^0[3][1,2,3] \\ z_2^0[0,1,2] = z_2^0[0], k^1[52{-}55] \oplus x_3^1[1], k^1[8, 9{-}11] \oplus x_0^1[2] \\ z_3^0[0,1,3] = k^1[48, 49{-}51] \oplus x_3^1[0], k^1[4{-}7] \oplus x_0^1[1], k^1[44{-}47] \oplus x_2^1[3] \end{cases}$$

6: Compute backwards (using a query to the block cipher) to $z^9$; compute $z_0^8, z_2^8, z_3^8$
7: Guess $z_0^7[1], z_1^7[0], z_2^7[3]$ ▷ 12 bits of state guess
8: Deduce the entire $z_0^7, z_1^7, z_2^7$
9: **Return** the following values:
▷ They correspond to the right hand sides of matching equations

Equation 22: $L_2'(k^1[9{-}12], k^1[49{-}52], k^1[53{-}56], (0|k^1[30,31]|0))$
Equation 23: $L_3'(k^1[9{-}12], k^1[49{-}52], k^1[53{-}56], (0|k^1[30,31]|0))$
Equation 24: $L_4'(k^1[9{-}12], k^1[49{-}52], k^1[53{-}56], (0|k^1[30,31]|0))$
Equation 25: $L_5'((k^1[31]|0|0|0) \oplus x_1^7[0], k^1[51{-}54] \oplus x_2^7[1], k^1[27{-}29, 30] \oplus x_0^7[3])$
Equation 26: $L_6'(k^1[47,48,49,50] \oplus x_2^7[0], k^1[23{-}26] \oplus x_0^7[2], k^1[43{-}46] \oplus x_1^7[3])$
Equation 27: $L_7'(k^1[19{-}22] \oplus x_0^7[1], (0|k^1[40{-}42]) \oplus x_1^7[2], (0|k^1[60{-}62]) \oplus x_2^7[3])$
Equation 28: $S(x_1^8[0]) \oplus L_8(x_0^9[3], x_1^9[0], (k^1[56]|0|0|0) \oplus x_2^9[1], (k^1[12]|0|0|0) \oplus x_3^9[2])$
Equation 28: $S(x_1^8[1]) \oplus L_9(x_0^9[3], x_1^9[0], (k^1[56]|0|0|0) \oplus x_2^9[1], (k^1[12]|0|0|0) \oplus x_3^9[2])$
Equation 28: $y_1^8[2] \oplus L_{10}(x_0^9[3], x_1^9[0], (k^1[56]|0|0|0) \oplus x_2^9[1], (k^1[12]|0|0|0) \oplus x_3^9[2])$

10: **end procedure**

For 1k-AES, we are able to attack 7 rounds (without the final linear layer) with the path of Figure 9. Recall that each cell $x_j^i$ represents the column $j$ of the state at the end of round $i$. We start by guessing the values of the 11 key bytes $k_{0,1,2,3,4,6,7,8,9,10,11}$. In the forward list, we guess the values of $k_{5,13,14,15}$ (4 bytes). In the backward list, we guess the value of $k_{12}$ (one byte).

We fix the 16 **green** $\leftrightarrow$ bytes on the figure, which contain 12 bytes of the plaintext, and 4 bytes of the state after one round. The **forward** ▼ list contains $2^{4\times 8}$ elements, as it depends only on the key bytes. The **backward** ▲ list also contains $2^{4\times 8}$ elements, as 3 state bytes need to be guessed. There are 4 bytes of matching between them, so the merged list is also of size $2^{4\times 8}$. Thus the time, memory and data complexities are respectively $2^{11\times 8} \times 2^{4\times 8} = 2^{120}$, $2^{4\times 8} = 2^{32}$, $2^{4\times 8} = 2^{32}$. We also obtained an attack of complexity $2^{112}$ on this variant, but with a larger data complexity.

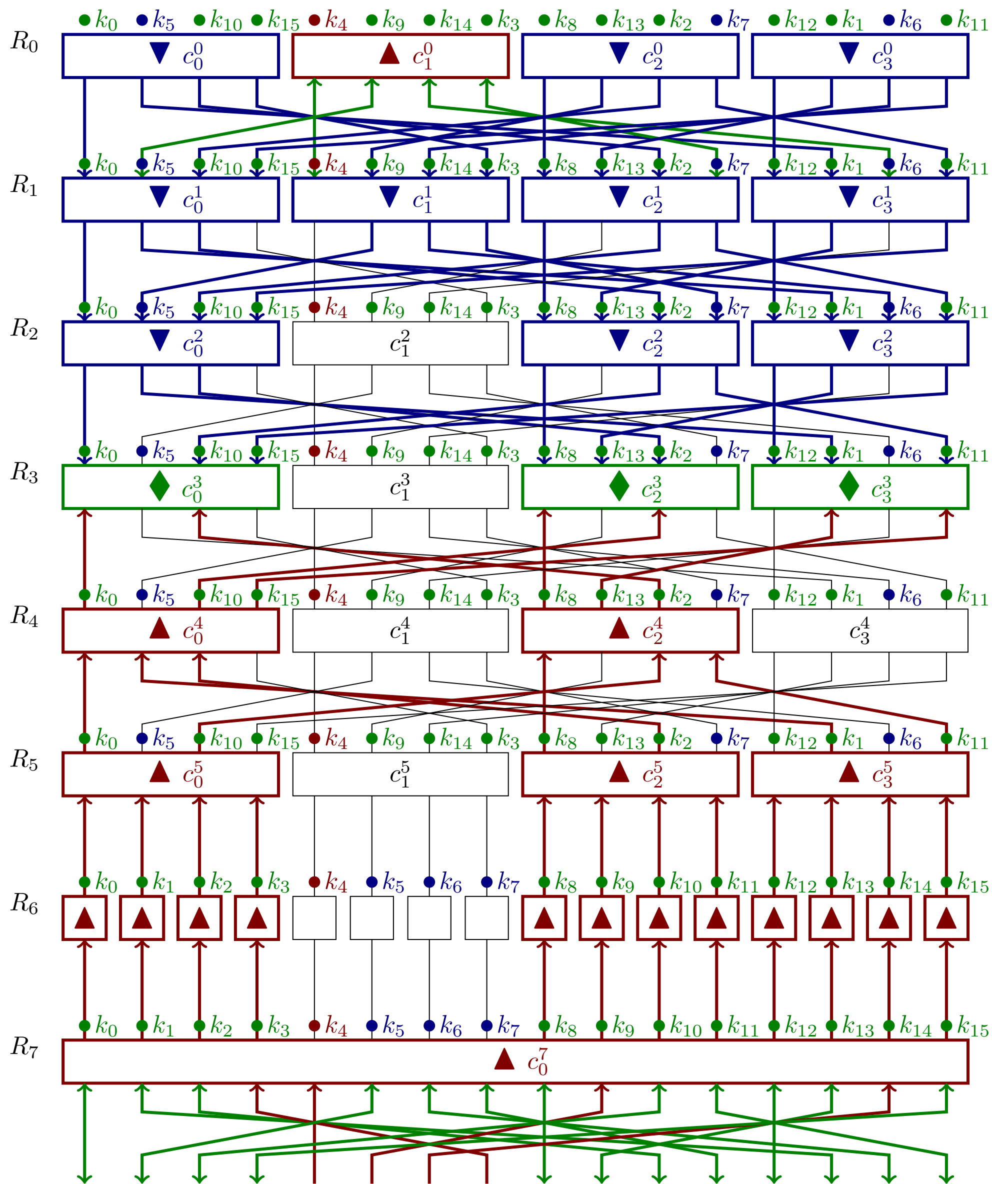


**Figure 9:** Path of a 7-round key-recovery attack on the 1k-AES structure.

While DS-MITM and Impossible Differential attacks would also reach 7 rounds, both would require a much larger memory and data complexity. For 2k-AES, we are able to attack 10 rounds, including the final linear layer. Compared to the DS-MITM attack on 10-round AES-256 [LJ16], our attack also benefits from a reduced memory complexity.

## C.2 Application to Saturnin-Hash

The path of our quantum pseudo-preimage attack on the SATURNIN-Hash compression function, reduced to 7 Super-rounds, is given in Figure 10. The forward list is of size $2^{16\times 4}$, the backward list is of size $2^{16\times 2}$, and in both cases this comes only from the key nibbles. A total of 16 state nibbles are fixed: 6 green edges between $R_3$ and $R_4$, 4 green edges between $R_4$ and $R_5$, and $3+3$ precomputed linear relations through $c_2^3$ and $c_3^3$.

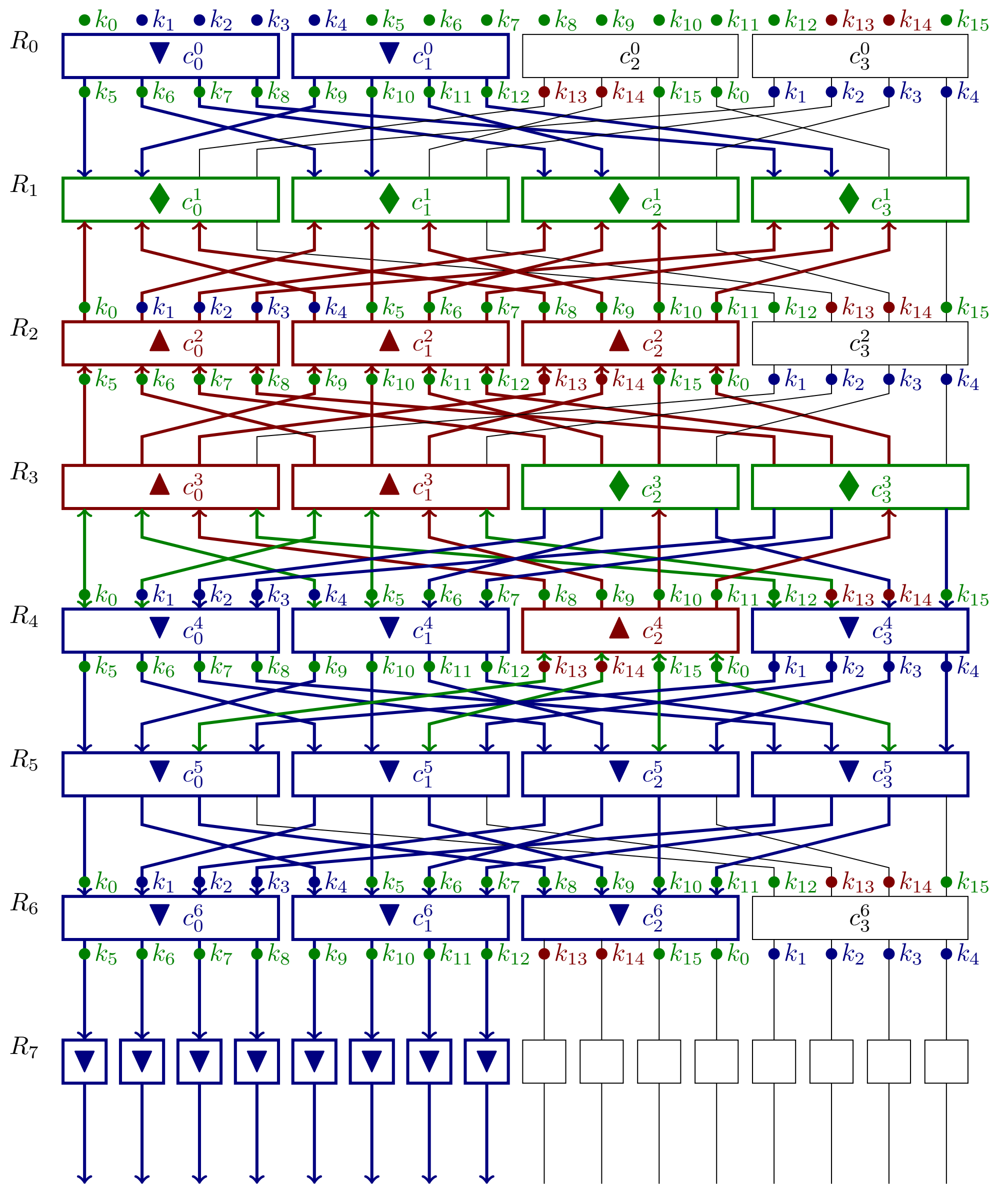


**Figure 10:** Path of a 7 Super-round quantum pseudo-preimage attack on SATURNIN-Hash.

## C.3 Application to Saturnin

The best classical key-recovery attack (in the single secret-key model) on Saturnin is the DS-MITM attack on 7.5 Super-rounds reported in [CDL+20], with time complexity $2^{244}$. This corresponds to 7 Super-rounds followed by another S-Box layer, and a key addition.

With our framework, we reach 6.5 Super-rounds with the path of Figure 11. The time complexity is $2^{248}$, and the data complexity is similar. The **forward** ▼ list contains half a key nibble ($k_{12}$) and two halves of state nibble guesses, so it has size $2^{24}$. The **backward** ▲ list contains 3.5 key nibbles ($k_{6,10,14}, k_{12}$). There are 2 nibble conditions for merging (one through $c_2^1$ and one through $c_0^4$), so the merged list is of size $2^{24}$.

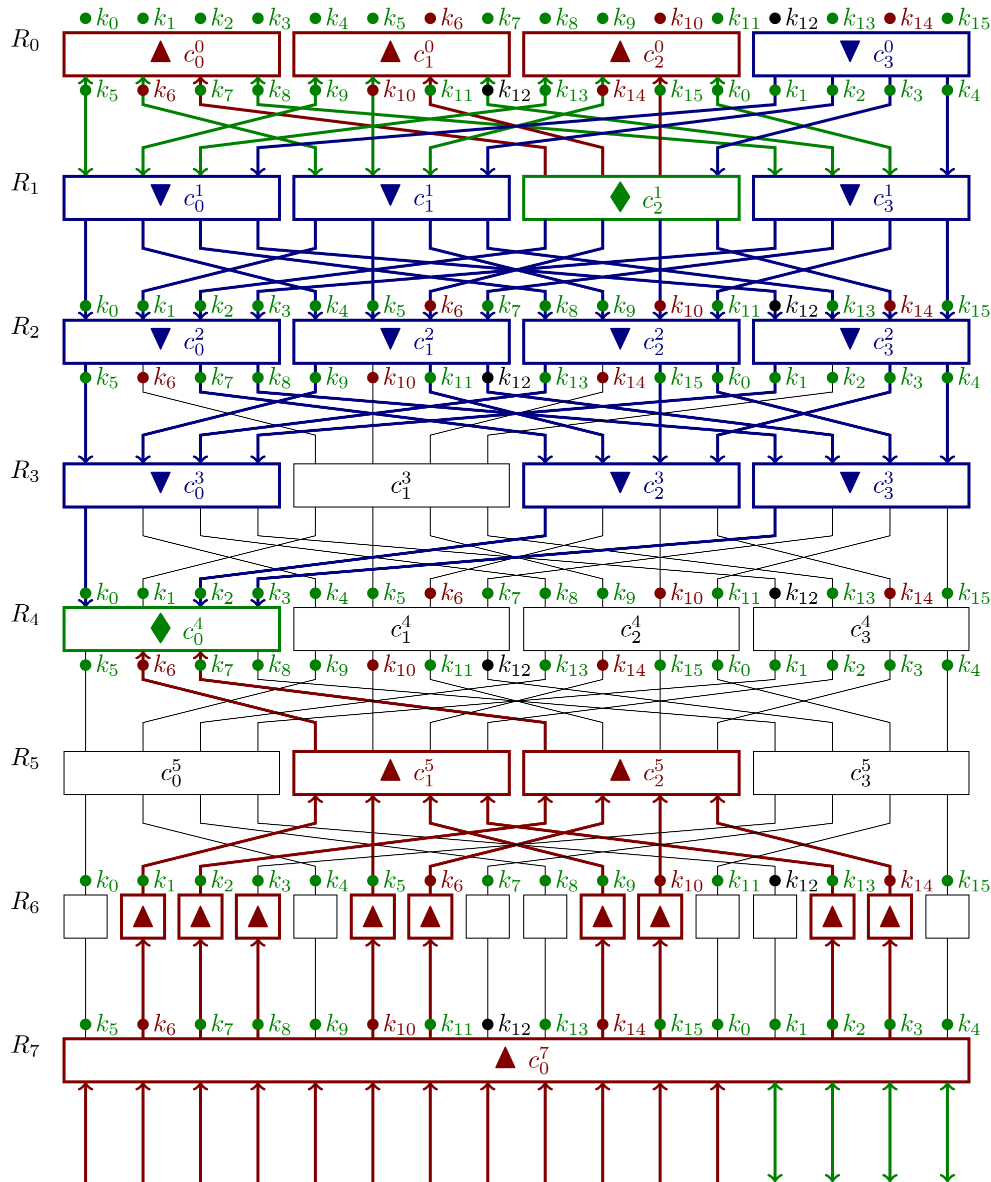


**Figure 11:** Path of a 6.5 Super-round key-recovery attack on Saturnin. The key nibble $k_{12}$ is half blue, half red.

For comparison, the best classical attack on 6.5 Super-round Saturnin would be a

Square attack, which initially targets 6-round AES [FKL+00]. State nibbles in Saturnin are twice the size of AES bytes, so the complexities are (roughly) squared. To accommodate the additional half Super-round, the time increases by a factor $2^{64}$ (corresponding to the guess of 4 key nibbles).

The best quantum key-recovery attack on Saturnin, as suggested in [CDL+20], is given by adapting the quantum Square attack of [BNS19]. On 6 Super-rounds it will reach a time complexity $2^{88}$, with roughly $2^{50}$ quantum memory and using $2^{70}$ classical chosen-plaintext queries (and classical memory). To reach 6.5 Super-rounds, the quantum time increases to approximately $2^{120}$ Saturnin evaluations.

In the quantum setting, our attack reaches a time complexity $2^{127.55}$ by Equation 4 which is almost at the generic limit of $2^{128}$ (roughly corresponding to a Grover search with large probability of success).

**Grover-meet-Simon Attack.** While 6 full Super-rounds are unreachable, we find a 5.5 Super-round Grover-meet-Simon attack which guesses all key nibbles except 3. By Equation 5, it has a complexity: $\frac{\pi}{2}2^{104} \times 48 \simeq 2^{110.24}$. Furthermore, the data complexity is only of $2^{64}$, meaning that we can use Q1 queries and store them in a QRACM of size $2^{64}$.

## C.4 Application to Haraka-v2

Haraka-v2 is a small-range hash function defined in [KLMR16] in two variants: Haraka-256 and Haraka-512, with respectively 256 and 512 bits of internal state. Both variants hash an input of respectively 256 and 512 bits using an AES-based permutation followed by a feedforward, and a truncation for Haraka-512.

The permutation is made in both cases of 5 rounds. Each round applies two AES rounds on the substates, followed by a MIX layer, which permutes the columns between substates.

Bao et al. [BDG+21] showed an attack on 4.5-round Haraka-256 and 5.5-round (extended) Haraka-512. The latter was subsequently improved in [SS22a] by reducing the memory used.

Since our model subsumes the one of [SS22a], we verify that it finds the same attacks on Haraka. Moreover, we were able to improve the attack on Haraka-512 and to devise an attack on 6.5-rounds Haraka-512. This shows that matching in many rounds (instead of a single round as in [BDG+21]) gives an advantage even greater than previously estimated. Note that both results were already covered by the previous model. Yet these attacks were not found in [SS22a], perhaps due to additional constraints used in [SS22a] to limit the search space. In our work, we used the Gurobi solver instead of SCIP which made the search more efficient.

## C.5 Application to Gift-64

Gift-64 [BPP+17] is a lightweight block cipher aiming at optimizing the Present design strategy, with a state of 64 bits, S-Boxes of 4 bits, and a linear layer which is a simple bit permutation. Gift-64 has a key of 128 bits.

The authors proposed a MITM key-recovery attack on 15 rounds, which was optimized by Sasaki [Sas18] using an automatic search based on MILP. We checked that in GAD mode, our model recovers the time ($2^{112}$) and memory ($2^{16}$) complexities of Sasaki's attack.

We also find a Parallel MITM attack with a slightly improved complexity. The path of this attack is actually the same as a Grover-meet-Simon attack that guesses 108 key bits, and has thus a complexity $\frac{\pi}{2}2^{54} \times 20 \simeq 2^{58.97}$ block cipher calls, using the same amount of Q2 queries. It only requires a small number of qubits.

## C.6 Application to Pipo-128

On 11-round Fly and 10-round Pipo-128, we use a path that fixes 105 key bits, that is, all of $k^0$ except 8 bits, and all of $k^1$ except 7 forward bits and 8 backward bits.

All lists are of size $2^{20}$. In the forward list, besides the 7 key bits, we need 8 bits of state guesses at round 4 and 5 bits at round 6. In the backward list, we need 4 bits at round 7. Then, because we know 4 backward cells at round 7 and 5 forward cells at round 6, there is $4 \times 4 = 20$ bits of matching between them, so the merged list is also of size $2^{20}$.

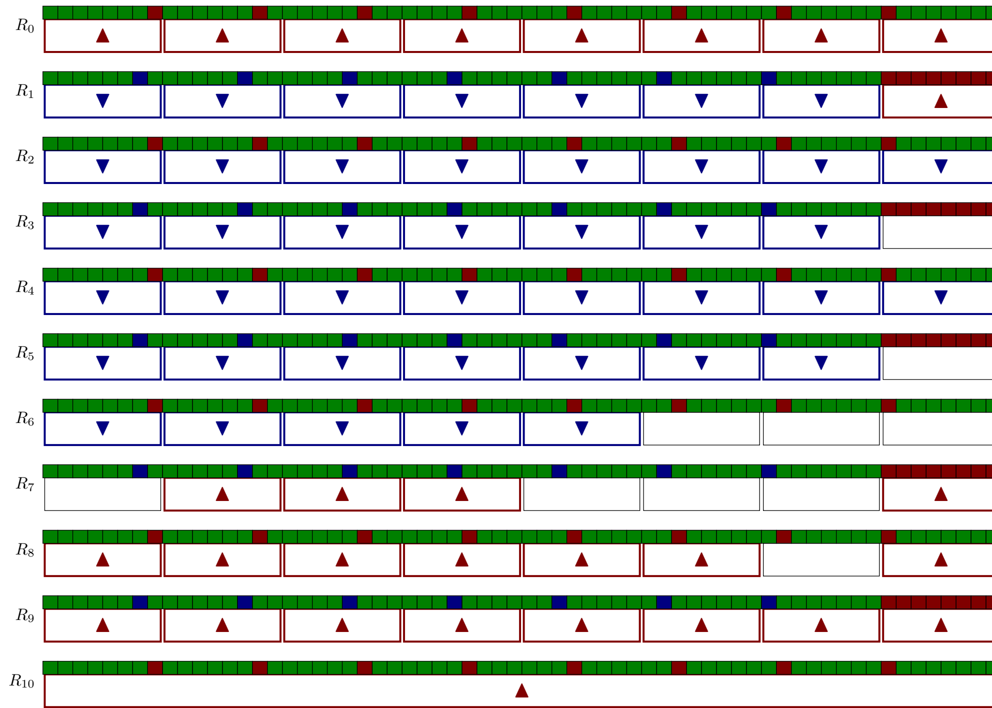


**Figure 12:** Path of the 10-round attack on Pipo-128.

A search for quantum attacks does not yield the same results. We find valid paths when removing two rounds. However, for Fly the number of rounds reached is smaller than quantum linear attacks [Sch23] and for Pipo-128 this is only as good.