Autoguess: A Tool for Finding Guess-and-Determine Attacks and Key Bridges [⋆]

Hosein Hadipour(✉) and Maria Eichlseder

Graz University of Technology, Graz, Austria
hsn.hadipour@gmail.com, maria.eichlseder@iaik.tugraz.at

Abstract. The guess-and-determine technique is one of the most widely used techniques in cryptanalysis to recover unknown variables in a given system of relations. In such attacks, a subset of the unknown variables is guessed such that the remaining unknowns can be deduced using the information from the guessed variables and the given relations. This idea can be applied in various areas of cryptanalysis such as finding the internal state of stream ciphers when a sufficient amount of output data is available, or recovering the internal state and the secret key of a block cipher from very few known plaintexts. Another important application is the key-bridging technique in key-recovery attacks on block ciphers, where the attacker aims to find the minimum number of required sub-key guesses to deduce all involved sub-keys via the key schedule. Since the complexity of the guess-and-determine technique directly depends on the number of guessed variables, it is essential to find the smallest possible guess basis, i.e., the subset of guessed variables from which the remaining variables can be deduced. In this paper, we present *Autoguess*, an easy-to-use general tool to search for a minimal guess basis. We propose several new modeling techniques to harness SAT/SMT, MILP, and Gröbner basis solvers. We demonstrate their usefulness in guess-and-determine attacks on stream ciphers and block ciphers, as well as finding key-bridges in key recovery attacks on block ciphers. Moreover, integrating our CP models for the key-bridging technique into the previous CP-based frameworks to search for distinguishers, we propose a unified and general CP model to search for key recovery friendly distinguishers which supports both linear and nonlinear key schedules.

Keywords: Lightweight block cipher · Guess & Determine · Key-Bridging · CP · MILP · SMT · SAT · Gröbner basis

1 Introduction

The practical security of symmetric-key cryptographic primitives with respect to known attacks is ensured by extensive cryptanalysis. There is a wide variety of different cryptanalytic techniques, including differential cryptanalysis, linear

[⋆] This is the full version of our paper accepted to ACNS 2022: https://doi.org/10.1007/978-3-031-09234-3_12 cryptanalysis, integral cryptanalysis (based on division properties), cube attacks, and more. Many of these cryptanalytic techniques involve tracing the propagation of certain cryptographic properties at the bit-level, which can be highly nontrivial. From a designer's perspective, to design a single primitive requires the analysis with all these known techniques. As a consequence, the design and cryptanalysis of symmetric-key primitives is a time-consuming and potentially error-prone process. Therefore, it is of significant importance for the community to develop automatic methods and tools.

One of the most widely used techniques in cryptanalysis is the guess-and-determine (GD) technique, especially when only low amounts of data are available to the attacker. Guess-and-determine is a general technique to recover the unknown variables in a given system of relations on a set of variables: A subset of the unknown variables is guessed such that the remaining unknowns can be deduced using the information from the guessed variables. The correctness of the guesses also can be checked using the given relations since it is assumed that the incorrect guesses yield inconsistency.

The idea can be used in various areas of cryptanalysis. For instance, the guess-and-determine technique can be applied to find all or part of the internal state of a stream cipher when a sufficient amount of output data is available. In the same way, the idea can be used to recover the internal state as well as the secret key of a block cipher when some known or chosen plaintext/ciphertext pairs are available. Another important application is the key-bridging technique in key recovery attacks on block ciphers, where the attacker aims to find the involved sub-keys based on the relations induced by the key schedule. Beyond these cryptanalytic uses, the guess-and-determine technique also finds application in a broader mathematical context, for example in its links to uniquely restricted matching problems in graph theory [45]. In these applications, the complexity of the guess-and-determine technique is directly dependent on the number of guessed variables. It is thus essential to find the smallest possible subset of guessed variables from which the remaining variables can be determined efficiently.

In this paper, we provide a general tool to search for a suitable set of guessed variables with minimum size in the guess-and-determine technique. This tool allows designers of symmetric-key primitives to easily and thoroughly analyze their designs from the guess-and-determine attack point of view. In addition to general guess-and-determine attacks on block and stream ciphers, our tool can help designers to optimize their key schedule algorithms with respect to the key-bridging technique.

Our contribution. Our contributions can be summarized as follows (see also Table 1):

1. We present *Autoguess*, an easy-to-use open-source tool to automate guess-and-determine attacks as well as the key-bridging technique. Adapting the method introduced by Cen *et al.* [14] to formulate guess-and-determine attack as an MILP problem, we introduce new encodings in CP and SAT/SMT which achieve a better performance compared to MILP [14] in many cases, particularly when searching for feasible solutions. In contrast to previous mod-

els [14, 21] where all variables should be deduced from the guessed variables, our reformulation takes an arbitrary subset of variables as the target variables into account. This enables us to extend the application to the key-bridging technique, where only an arbitrary subset of variables needs to be deduced. Besides, we adapt the method introduced by Danner *et al.* [21] to translate guess-and-determine attacks to the problem of computing the Gröbner basis of a Boolean ideal, and extend it for key-bridging technique as well. As a consequence, Autoguess integrates a wide range of CP/SMT/SAT/MILP solvers as well as the Gröbner basis algorithm to automate guess-and-determine attacks and the key-bridging technique.

2. Applying our tool to search for key bridges in word-oriented block ciphers, we improve the related-tweakey zero-correlation attacks on `SKINNY`-64-128 and `SKINNY`-64-192 from ToSC 2019 [2], by deducing one nibble of involved sub-keys for free. Utilizing Autoguess to search for key-bridges in bit-oriented block ciphers with nonlinear keyschedule, we also reduce the time complexity of the analysis phase in linear attack on 26-round `PRESENT`-80 presented at EUROCRYPT 2020 [38] from 2^{65} to 2^{64}.
3. To show the application of our tool in the analysis of stream ciphers, we use it to reduce the computational complexity of the guess-and-determine attack on `ZUC` from 2^{392} [36] to 2^{390} while using the same amount of 9 keystream output words.
4. We show that our tool can automatically re-discover many of the best results obtained with the key-bridging technique, which previously had to be generated either manually or with dedicated, cipher-specific tools. We successfully automatically re-discovered the Demirci-Selçuk meet-in-the-middle attack on `SKINNY`-128-384 [15] and the integral attack on 24-round `LBlock` [19]. To show the versatility of our tool, we also used it for finding low-data-complexity attacks on block ciphers. More precisely, we used it to find guess-and-determine attacks on `AES`, `Khudra`, `CRAFT`, and `SKINNY`. For example concerning `AES`, we could rediscover the best previous GD attack on 3 rounds with data complexity of merely one known plaintext/ciphertext pair.
5. Lastly, we show that our CP-based approaches for the key-bridging technique are consistent with the previous CP-based frameworks to search for distinguishers. Thanks to this consistency, we integrate it into the the previous CP-based frameworks for automatic search of distinguishers to build a general CP-model to find the key recovery friendly distinguishers taking the key-bridging into account for both linear and nonlinear key schedules. To show the usefulness of this new method, we could improve the memory complexity of the best previous $\mathcal{DS}$-MITM attack on 20-rounds of `TWINE`-80 by a factor of 2^{20}. We also utilized this new framework to find the $\mathcal{DS}$-MITM attacks on `SKINNY`-64-128, `SKINNY`-64-192 and `SKINNY`-128-256 for the first time.

Software. We have provided an open-source tool Autoguess, implementing our methods, which is publicly available via the following link:

`https://github.com/hadipourh/autoguess`

Table 1: Summary of Our Attacks on SKINNY-128-256, SKINNY-64-192, SKINNY-64-128 and TWINE-80, where $\mathcal{DS}$-MITM denote Demirci-Selçuk Meet-in-the-Middle cryptanalysis and ST stands for single-tweakey setting.

Cipher	#Rounds	Data	Memory	Time	Attack	Setting	Reference
SKINNY-128-256	19	2^{96} CP	$2^{210.99}$	$2^{238.26}$	$\mathcal{DS}$-MITM	ST	Section 9.2
SKINNY-64-192	21	2^{60} CP	$2^{133.99}$	$2^{186.63}$	$\mathcal{DS}$-MITM	ST	Section 9.3
SKINNY-64-128	18	2^{32} CP	$2^{61.91}$	$2^{126.32}$	$\mathcal{DS}$-MITM	ST	Section 9.4
TWINE-80	20	2^{32} CP	$\mathbf{2^{62.91}}$	$2^{76.92}$	$\mathcal{DS}$-MITM	-	Section 9.1
TWINE-80	20	2^{32} CP	$2^{82.91}$	$2^{77.44}$	$\mathcal{DS}$-MITM	-	[62]

Outline. In Section 2, we recall the preliminaries on guess-and-determine attacks and the key-bridging technique. In Section 3, we propose the constraint programming model of these two techniques, and in Section 4, we discuss an alternative model using Gröbner bases. In Section 5, we introduce our tool Autoguess with its preprocessing and early-abort techniques. We apply it to find key bridges for different ciphers in Section 6 as well as guess-and-determine attacks on block ciphers in Section 7 and stream ciphers in Section 8. Finally, we provide a discussion in Section 10 and conclude in Section 11.

2 Preliminaries

In this section, we introduce the notation used in this paper and provide a brief overview of the cryptanalytic background, including the guess-and-determine and key-bridging techniques, as well as modeling connection relations.

2.1 Notation

Table 2 summarizes the used notation throughout this paper.

2.2 Guess-and-Determine Technique

The guess-and-determine technique is a general method to solve a system of equations, given as a set of variables linked by relations. In this method, the values of a subset of the variables are guessed first. Next, using the relations, one may find the values of a subset of the remaining unknown variables, which is called knowledge propagation. If all of the remaining unknown variables are determined from the guessed variables, we call the set of guessed variables a guess basis [11]. In other words, a guess basis is a set of variables such that by knowing their values, we can recover the remaining unknown variables.

For a given guess basis, we can enumerate all possible values for its variables, derive all other variables, and verify whether the result satisfies all relations. The solution set for the system of equations consists of those guesses for which all relations are satisfied. Hence, the complexity is roughly equal to the number of

Table 2: Notation.

$\neg$	bit-wise NOT
$\lll i$, $\ggg i$	left rotation and right rotation by i bits, respectively
$\|\|$	concatenation
$\wedge$	bit-wise And
$\vee$	bit-wise Or
$\oplus$	bit-wise Xor
$\boxplus$	modular addition
$\boxminus$	modular subtraction
$x \Rightarrow y$	y can be deduced from x
`BVZExt`(x, n)	zero-extension of x by n bits: $\overbrace{0\|\|\cdots\|\|0}^{n}\|\|x$
`BVAdd`(x, y)	bit-vector addition of bit-vectors x, and y
`BVULE`(x, y)	unsigned less than or equal comparison of two bit-vectors x, and y
$i \sim j$	successive numbers from i to j

possible assignments for the guess basis. Therefore, the main goal is finding a guess basis of minimal size.

For systems of equations obtained from cryptographic primitives, the guess-and-determine technique is often used when data is very scarce, and statistical attacks are therefore impossible. The main challenges of a guess-and-determine attack are to find a suitable guess basis and to effectively propagate knowledge.

Guess-and-determine attacks were among the first ideas to recover internal state of stream ciphers. [63] applied a divide-and-conquer attack to detect and recover the internal states of LFSR-based ciphers. The attack was successfully applied to a number of ciphers, including `A5/1` [35], `RC4` [46], `SOBER` [4] and `SNOW1` [40] constituting GD attacks on word-oriented stream ciphers. Since the complexity of these attacks depends crucially on the size of the guess basis, several improvements and approaches to minimize the number of guesses have been proposed. For instance, Ahmadi and Ehglidos [1] proposed a heuristic approach based on dynamic programming to automatically find guess-and-determine attack for classes of stream ciphers. As another example, [70] uses the idea of SAT solver and Monte-Carlo Optimization algorithms to find the guessed variables.

2.3 Key-Bridging Technique

Attacks on block ciphers can typically be divided into two parts: A distinguishing part and a key recovery part. In the distinguisher part, the attacker finds a certain property to distinguish r_2 rounds of the cipher from a pseudorandom permutation. Then, they add r_1 and r_3 rounds before and after the distinguished part, respectively, to mount a key-recovery attack on $r = r_1 + r_2 + r_3$ rounds. In the key recovery part, the attacker guesses some key bits involved in first r_1 and last r_3 rounds to find the internal states corresponding to the input and output of the r_2 middle rounds, for a particular set of messages, to see whether a certain property for the middle part is satisfied.

The guessed key-bits may have some relations due to the key schedule, even though they are separated by many rounds. The main goal of the key-bridging technique is to find these relations and reduce the complexity of key recovery by deducing some sub-keys from the other sub-keys. In other words, in the key-bridging technique, we look for a minimal set of guessed sub-key bits which is sufficient to determine all involved sub-key bits in the key recovery attack.

The key schedule of block ciphers is typically much simpler than the main round function due to restrictions in memory and area consumption. This is a potential source of weakness that can be exploited in attacks such as the key-bridging technique. Key-bridging was introduced by Dunkelman *et al.* at ASIACRYPT 2011 [26], where in their attack on 8 rounds of `AES`-192, they provide a surprisingly long bridge between two sub-keys which are separated by 8 key schedule rounds. At EUROCRYPT 2013, Derbez *et al.* improved the attack on 8 rounds of `AES`-192, in which they used key-briding technique to lower the time complexity of the attack. At FSE 2015, Biryukov *et al.* applied key-bridging in meet-in-the-middle and impossible differential attacks on 25 rounds of `TWINE`-128 [10]. One of the most interesting works concerning key-bridging is an automatic method to search for key-bridging, introduced by Lin *et al.* at FSE 2016 [49]. They apply their approach to impossible differential and multidimensional zero-correlation linear attacks on 23-round `LBlock`, 23-round `TWINE`-80, and 25-round `TWINE`-128. However, their method is not sufficiently general to handle all cryptographic operations such as modular addition, subtraction, and multiplication. Besides, it merely determines an upper bound for the number of solutions by extracting some relations between the key variables and does not specify the guessed variables as well as the determination flow and hence is not applicable to search for GD attacks on stream ciphers. Moreover, the tool presented in [49] is based on a dedicated linear algebraic method and hence is not consistent with the CP-based approaches to search for distinguishers, whereas as we will show in Section 9, our CP-based approaches for key-bridging technique can be merged into the previous CP-based tools for finding distinguishers.

2.4 Connection Relations

In order to describe the guess-and-determine technique in a systematic way, we define two types of connection relations [14].

Definition 1 (Implication Relation) *Let $x_0, \ldots, x_{n-1}, y$ denote some variables. If y can be uniquely determined from $x_0, \ldots, x_{n-1}$, we say that $x_0, \ldots, x_{n-1}, y$ have an implication relation, denoted as*

$$r : x_0, \ldots, x_{n-1} \Rightarrow y \, .$$

We denote the left- and right-hand sides of this relation r as $\texttt{LHS(r)} = \{x_0, \ldots, x_{n-1}\}$ and $\texttt{RHS}(r) = \{y\}$, respectively.

Example 1 *Let $(y_3, y_2, y_1, y_0) = S(x_3, x_2, x_1, x_0)$ denote a 4-bit S-box. Four implication relations can be derived as follows:*

$$\forall \; 0 \leq i \leq 3 : x_0, x_1, x_2, x_3 \Rightarrow y_i \, .$$

If S is a bijective function, then four extra implication relations $y_0, y_1, y_2, y_3 \Rightarrow x_j$ for all $0 \leq j \leq 3$ can be derived as well.

Definition 2 (Symmetric relation) *Let $x_0, \ldots, x_{n-1}$ denote n variables. We say they have a symmetric relation if and only if each variable x_i can be uniquely deduced when the remaining $n-1$ variables are all known, denoted as*

$$r : [x_0, x_2, \ldots, x_{n-1}] .$$

The number of distinct variables appearing in r is defined as the length $|r|$ of r.

Example 2 *Let $F : \mathbb{F}_2^{32} \rightarrow \mathbb{F}_2^{32}$ be a bijective function and $x, y, z, k \in \mathbb{F}_2^{32}$, where $z = F(x \boxplus k) \oplus y$. Then, by knowing the value of three out of four variables x, y, z, k in this relation, the value of the remaining one can be uniquely determined, x, y, z, and k have a symmetric relation, which is denoted by $[x, y, z, k]$.*

Although a symmetric relation involving n variable can be described by n implication relations, we prefer to use this compact notation when it is possible.

2.5 From a System of Equations to a System of Connection Relations

In this paper, we assume that the given system of equations can be described via a combination of implication and symmetric relations. This assumption does not impose significant limitations since equations for many cryptographic systems can be described completely using a combination of implication or symmetric connection relations. For instance, some rules are listed below to convert a system of equations to a system of connection relations.

Proposition 1 (Xor) *If $y = \bigoplus_{i=0}^{n-1} x_i$, where $x_0, x_1 \ldots, x_{n-1}, y \in \mathbb{F}_2^n$, then $[x_0, \ldots, x_{n-1}, y]$.*

Proposition 2 (And) *If $y = \bigwedge_{i=0}^{n-1} x_i$, where $x_0, x_1 \ldots, x_{n-1}, y \in \mathbb{F}_2^n$, then $x_0, \ldots, x_{n-1} \Rightarrow y$.*

Proposition 3 (Modular Addition) *If $y = \boxplus_0^{n-1} x_i$, then $[x_0, \ldots, x_{n-1}, y]$, where $x_0, x_1 \ldots, x_{n-1}, y \in \mathbb{F}_2^n$.*

Proposition 4 (Bijective Function) *If $F : \mathbb{F}_2^m \rightarrow \mathbb{F}_2^n$ is a bijective function and $y = F(x)$, then $[x, y]$.*

Proposition 5 (Split/Concatenation) *Let $x = x_0 || \ldots || x_{n-1}$. The following connection relations can be used to model this operation:*

$$x_0, \ldots, x_{n-1} \Rightarrow x; \; \forall \; 0 \leq i \leq n-1 : x \Rightarrow x_i.$$

Proposition 6 (Elimination rule) *If $[x_0, \ldots, x_{n-1}, x] \wedge [x, y_0, \ldots, y_{n-1}]$, then:*

$$[x_0, \ldots, x_{n-1}, y_0, \ldots, y_{n-1}].$$

Using a combination of the above rules, more complex equations can be modeled as well.

Example 3 *Let $y_1 = ((x_0 \ggg 8) \boxplus y_0) \oplus (y_0 \lll 3)$, where $x_0, y_0, y_1 \in \mathbb{F}_2^{64}$. By introducing some new variables, and then using the elimination rule, this equation can be modeled via the symmetric relation $[x_0, y_0, y_1]$.*

2.6 A Naive Approach for Guess and Determine Attack

Assume that there is a system of connection relations involving n unknown variables for which we are looking for a guess basis of minimum size. A naive approach to find a minimal guess basis is the exhaustive search method which starts from $k = 1$, checking all possible subsets of size k where $1 \leq k \leq n$, to discover a minimal subset holding the property of guess basis. To check whether a subset K of size k can be a guess basis, assuming that all variables in K are known, through the given connection relations we propagate the knowledge to obtain the new set of known variables $Y = \texttt{Propagate}(K)$, where `Propagate` performs the knowledge propagation (see Algorithm 4 in Appendix A). A minimal guess basis is found as soon as a set $\mathbb{G}$ is found for which $|\texttt{Propagate}(\mathbb{G})| = n$ (Algorithm 5, Appendix).

Consequently, the complexity of the exhaustive search for a guess basis of size less than or equal to m (if it exists) is roughly equal to $\sum_{k=1}^{m} \binom{n}{k}$, which is exponential with respect to both n and m. Therefore, this approach is infeasible when m or n are large enough.

3 Constraint Programming Model of Guess-and-Determine and Key-Bridging Techniques

In this section, we propose an SMT/SAT model of guess-and-determine attacks and the key-bridging technique. Our model is inspired by the method introduced by Cen et al. [14] to convert guess-and-determine attacks to an MILP problem. Modeling these techniques as SMT/SAT problems is not only more straightforward than MILP, but also achieves a better performance in some cases, especially when we want to find only a feasible solution. As a new cryptographic application, we extend our approach to automatic search for key-bridging technique for the first time.

3.1 From GD and Key-Bridging to Constraint Programming Model

Given a system of connection relations $(X, \mathcal{R})$, where $X = \{x_0, \ldots, x_{n-1}\}$, and $\mathcal{R} = \{r_0, \ldots, r_{m-1}\}$, Cen *et al.* [14] introduced a method to formulate the problem of searching for a minimal guess basis for X to an MILP problem. Here, the CP and SMT/SAT alternative of this method is proposed and as a new cryptographic application it is used to search for key-bridging technique as well.

Two main challenges of the guess-and-determine technique are knowledge propagation and finding a minimal guess basis [11]. Finding a minimal guess basis is an optimization problem. However, using a standard technique it can be transformed into the sequence of decision problems in each of which, it is answered that whether the sets of specific sizes can be a guess basis. One can find a minimal guess basis by decreasing the explored sizes in the sequence of decision problems. Therefore, the first and more critical step is modeling the decision problem of whether sets of certain sizes can be a guess basis for which one has to model the knowledge propagation.

The knowledge propagation is a procedural method in which the only considered (given) property of variables is whether a variable is known or unknown. However, as it is depicted in Algorithm 4, during the knowledge propagation one variable which is unknown at the current step may be determined from the known variables at the next step. Hence, to model whether a variable is known or unknown, more than one decision variable is needed.

Let $(X, \mathcal{R})$ be a system of connection relations, where $X = \{x_0, \ldots, x_{n-1}\}$ and $\mathcal{R} = \{r_0, \ldots, r_{m-1}\}$. Assuming that a subset of variables such as $K_0 \subseteq X$ is initially known, the known/unknown status of each single variable $x \in X$ in each step can be represented by a new binary decision variable, the *state variable*:

Definition 3 (State variables) *For a given system of connection relations $(X, \mathcal{R})$, where $X = \{x_0, \ldots, x_{n-1}\}$, to represent the status of variables in terms of being known or unknown in jth step of knowledge propagation, a set of binary decision variables $S_j = \{x_{0,j}, \ldots, x_{n-1,j}\}$ is defined such that $x_{i,j} = 1$, if and only if x_i is known at step j of knowledge propagation and $x_{i,j} = 0$ otherwise, where $0 \le i \le n-1$ and $j \in \mathbb{Z}_{\ge 0}$.*

Therefore, for a given initial subset $K_0 \subseteq X$ of known variables, the knowledge propagation can be represented using the following chain, where $S_j = \{x_{0,j}, \ldots, x_{n-1,j}\}$ represents the status of variables in terms of being known or unknown at jth step of knowledge propagation:

$$S_0 \to S_1 \to \cdots \to S_j \to \cdots .$$

Given that a variable can be involved in more than one connection relation, to link each variable to its corresponding relations, a new type of binary decision variable called *path variable* is defined as follows.

Definition 4 (Path variables) *Let $(X, \mathcal{R})$ be a system of connection relations with $R = \{r_0, \ldots, r_{m-1}\}$. Assume $x_i \in X$ appears in λ relations $\{r^i_0, \ldots, r^i_{\lambda-1}\}$, where for each $0 \le k \le \lambda - 1$, r^i_k is either a symmetric relation or an implication relation with $x_i \in \texttt{RHS}(r^i_k)$. Then, for each step j of knowledge propagation, λ new binary decision variables $\texttt{Path}(x_{i,j}) := \{x_{i,j,k} : 0 \le k \le \lambda - 1\}$ are defined as follows:*

$$x_{i,j,k} = \begin{cases} 1 & x_i \text{ can be determined from the relation } r^i_k \text{ at step } j-1 \\ 0 & \text{otherwise,} \end{cases}$$

where $0 \le i \le n-1$ and $j \in \mathbb{Z}_{\ge 1}$. $\texttt{Path}(x_{i,j})$ is called the set of path variables corresponding to $x_i \in X$ at the jth step of knowledge propagation. For $j = 0$ and all $0 \le i \le n-1$, $\texttt{Path}(x_{i,0}) = \emptyset$.

Proposition 7 (Knowledge propagation) *Given a system of connection relations $(X, \mathcal{R})$, let $x_{i,j}$ and $\texttt{Path}(x_{i,j}) = \{x_{i,j,k} : 0 \le k \le \lambda - 1\}$ represent the state variable and set of path variables corresponding to $x_i \in X$ at jth step of knowledge propagation, respectively. Then, x_i at the jth step of knowledge propagation is known, i.e., $x_{i,j} = 1$, if and only if at least one of the following conditions holds:*

- ***Already known:*** *$x_{i,j-1} = 1$, i.e., x_i has been known since the previous steps,*
- ***Determined:*** *There exists $x_{i,j,k} \in \texttt{Path}(x_{i,j})$ such that $x_{i,j,k} = 1$, i.e., $x_{i,j}$ can be determined from the previously known variables.*

For a given system of connection relations $(X, \mathcal{R})$ and a subset of known variables, any assignment for the state and path variables satisfying the definitions of state and path variables as well as Proposition 7 corresponds to a valid knowledge propagation. For a valid assignment of state and path variables, let $K_j := \{x_{i,j} \in S_j : x_{i,j} = 1\}$. According to the first condition in Proposition 7, if $x_{i,j} = 1$, then for all $j' \geq j$, $x_{i,j'} = 1$, where $0 \leq i \leq n-1$, since a variable remains known after it becomes known once. As a consequence, $K_0 \to \cdots \to K_j \to \cdots$, where $j \in \mathbb{Z}_{\geq 0}$ is an ascending chain. On the other hand, the number of known variables is upper bounded by $|X| = n$. Therefore, according to the ascending chain condition, there exists a positive integer β such that $K_\beta = K_{\beta+1} = \cdots$.

While in the guess-and-determine technique one usually looks for a minimal guess basis to deduce all of the remaining variables, in the key-bridging technique we are looking for a minimal set of guessed variables to determine a certain subset of variables, which are the sub-keys involved in the key-recovery. Accordingly, we define the concept of guess basis for a subset $\mathcal{T} \subseteq X$ as the target variables, where $(X, \mathcal{R})$ is a system of connection relations.

Definition 5 (Guess basis) *Let $(X, \mathcal{R})$ be a system of connection relations and $\mathcal{T} \subseteq X$. The subset $K \subseteq X$ is called a guess basis for $\mathcal{T}$, if there exists some positive integer β such that all variables in $\mathcal{T}$ can be deduced from K after β steps of knowledge propagation.*

Using the following proposition, one can characterize the guess basis for a given system of connection relations and a subset of target variables.

Proposition 8 (Characterizing guess basis) *Let $(X, \mathcal{R})$ be a system of connection relations and $S_0 \to \cdots \to S_j \to \cdots$ be its corresponding chain of state variables. $K_0 \subseteq X$ is a guess basis for $\mathcal{T} \subseteq X$, if there exists a positive integer β, and an assignment of state and path variables for which the following conditions hold:*

- *For all $x_{i,0} \in S_0$, if $x_i \in K_0$, then $x_{i,0} = 1$, and $x_{i,0} = 0$ otherwise.*
- *The assignment satisfies the definition of state and path variables and Proposition 7.*
- *For all $x_{i,\beta} \in S_\beta$, if $x_i \in \mathcal{T}$, then $x_{i,\beta} = 1$, i.e., all target variables should be known in the final step of knowledge propagation.*

Encoding Using CP

Proposition 9 (Link from path to state variables in CP encoding) *Let $x_{i,j,k}$ be a path variable corresponding to the state variable $x_{i,j}$ in connection relation r^i_k. Assume that the variables of r^i_k are x_i and $x_{i_0}, \ldots, x_{i_{p-1}}$ for some $p \in \mathbb{Z}_{\geq 1}$.*

*Then, the link between $x_{i,j,k}$, and the state variables $x_{i_0,j-1}, \ldots, x_{i_{p-1},j-1}$, is encoded as follows, where $j \in \mathbb{Z}_{\geq 1}$, and r_k^i is either a symmetric relation, or an implication relation such that $x_i \in$ **RHS**(r_k^i):*

$$x_{i,j,k} = x_{i_0,j-1} \wedge \cdots \wedge x_{i_{p-1},j-1}.$$

Proposition 10 (Link from state to path variables in CP encoding) *Let* **Path**$(x_{i,j})$ *be the set of path variables $\{x_{i,j,k} : 0 \leq k \leq \lambda - 1\}$ corresponding to the state variable $x_{i,j}$. The link between $x_{i,j}$ and* **Path**$(x_{i,j})$ *can be encoded as follows:*

$$x_{i,j} = x_{i,j-1} \vee x_{i,j,0} \vee \cdots \vee x_{i,j,\lambda-1}.$$

For a given system of connection relations $(X, \mathcal{R})$, let $\mathcal{T} \subseteq X$ be the set of target variables for which we are looking for a minimal guess basis. To encode this problem into a CP model, we firstly consider a fixed positive integer value for β as the depth of knowledge propagation and then generate the state and path variables corresponding to β steps of knowledge propagation. Assume that the chain $S_0 \to \cdots \to S_\beta$, represents the knowledge propagation through β steps where $S_j = \{x_{i,j} : 0 \leq i \leq n-1\}$. Then, we set the objective function as follows:

$$\min \sum_{i=0}^{n-1} x_{i,0},$$

such that $x_{i,\beta} = 1$ for all $x_i \in \mathcal{T}$, and all CP constraints corresponding to links between the state and path variables (Proposition 9 and 10) are satisfied. If the constructed CP model is satisfiable, then the set $\mathcal{M} := \{x_i \in X : x_{i,0} = 1\}$ is a guess basis for $\mathcal{T}$. According to the ascending chain rule, the set $\mathcal{M}$ converges to a minimal guess basis for $\mathcal{T}$ when β is large enough. Algorithm 1 summarizes the steps required for CP encoding.

Encoding Using SMT Satisfiability Modulo Theories (SMT) refers to the decision problem of deciding whether a first-order logic formula is satisfiable with respect to combinations of background theories such as arithmetic, bit-vectors and arrays. SMT problems can be considered as a generalization of SAT problems, since they can express constraints on a higher abstraction level while SAT problems are only expressed in propositional (zero-order) logic. Most SMT solvers can not only determine the satisfiablity of a problem but also obtain a satisfying assignment [5,22,27,55]. In addition, some SMT solvers such as Z3 [22] can also be used to find a satisfying assignments that are optimal with respect to some objective functions. This allows applying SMT solvers not only for search problems, but also for optimization problems. Using these features, encoding knowledge propagation as SMT constraints is more intuitive compared to MILP encoding [14]. We will show in Section 6 that using the SMT solvers also gives a better performance in comparison to the MILP solvers in some cases, particularly when we look for only a feasible solution rather than solving an optimization

Algorithm 1: CP Encoding

```
Input: A system of connection relations $(X, \mathcal{R})$, where $X = \{x_0, \ldots, x_{n-1}\}$, a
       set of target variables $\mathcal{T} \subseteq X$, the depth $\beta \in \mathbb{Z}_{\geq 1}$ of knowledge
       propagation
Output: A sufficient subset $\mathbb{G} \subseteq X$ for $\mathcal{T}$
1  Initialize the dictionary Deriver, where $keys(\texttt{Deriver}) = X$;
2  for $i = 0 \to n - 1$ do
3      Deriver$[x_i] \leftarrow [\{x_i\}]$;
4      for $r \in \mathcal{R}$ do
5          if $r$ is a symmetric relation and $x_i \in r$ then
6              Deriver$[x_i] \leftarrow$ Deriver$[x_i] \cup [\{v \in r : v \neq x_i\}]$;
7          if $r$ is an implication relation and $x_i \in RHS(r)$ then
8              Deriver$[x_i] \leftarrow$ Deriver$[x_i] \cup [$LHS$(r)]$;
9  Declare an empty CP model $\mathcal{M}$;
10 for $j = 0 \to \beta - 1$ do
11     for $i = 0 \to n - 1$ do
12         $\mathcal{M}.var \leftarrow \{x_{i,j+1}\}$;
13         $\lambda \leftarrow |$Deriver$[x_i]|$;
14         for $k = 0 \to \lambda - 1$ do
15             Let Deriver$[x_i][k] = \{x_{i_0}, \ldots, x_{i_{p-1}}\}$;
16             $\mathcal{M}.var \leftarrow \{x_{i,j+1,k}\} \cup \{x_{i_0,j}, \ldots, x_{i_{p-1},j}\}$;
17             $\mathcal{M}.con \leftarrow x_{i,j+1,k} = \bigwedge_{l=0,\ldots,p-1} x_{i_l,j}$      ▷ Link path to the state
                 variables;
18         $\mathcal{M}.con \leftarrow x_{i,j+1} = \bigvee_{k=0,\ldots,\lambda-1} x_{i,j+1,k}$      ▷ Link state to the path
             variables;
19 for $x_i \in \mathcal{T}$ do
20     $\mathcal{M}.con \leftarrow x_{i,\beta} = 1$      ▷ Target variables must be known in final step ;
21 $\mathcal{M}.obj \leftarrow \min . \sum_{i=0}^{n-1} x_{i,0}$      ▷ Objective function ;
22 solution $\leftarrow \mathcal{M}.solve$      ▷ Call a CP solver;
23 return $\mathbb{G} = \{x_i \in X | x_{i,0} = 1\}$;
```

problem. Hence, we are motivated to utilize the SMT model alongside the MILP model in our tool.

We encode the links between the state and path variables into SMT constraints, using the bit-vector theory. An SMT problem in the bit-vector theory includes a set of bit-vector variables and a set of bit-vector constraints defined using logical (e.g., `Equals`, `NotEquals`, `Implies`, etc.) and bit-vector operations (e.g., $\oplus, \boxplus, \lll$, etc.).

Proposition 11 (Link from path to state variables in SMT encoding) *Let $x_{i,j,k}$ be a path variable corresponding to the state variable $x_{i,j}$ appearing in the connection relation r_k^i, and let $\{x_{i_0}, \ldots, x_{i_p}\}$ denote the variables included in r_k^i besides x_i. Then $x_{i,j,k} = 1$ if and only if x_i can be determined from r_k^i at step $j - 1$. This link between $x_{i,j,k}$ and the state variables $\{x_{i_0,j-1}, \ldots, x_{i_p,j-1}\}$ can* *be encoded as follows, where r_k^i is either a symmetric relation or an implication relation with $x_i \in$ RHS(r_k^i):*

$$\texttt{Equals}(x_{i,j,k}, x_{i_0,j-1} \wedge \cdots \wedge x_{i_p,j-1}).$$

Proposition 12 (Link from state to path variables in SMT encoding) *Let* Path$(x_{i,j}) = \{x_{i,j,k} : 0 \leq k \leq \lambda - 1\}$ *be the path variables corresponding to the state variable $x_{i,j}$. According to Proposition 7, $x_{i,j} = 1$ if and only if $x_{i,j-1} = 1$, or there exists $x_{i,j,k} \in$ Path$(x_{i,j})$ such that $x_{i,j,k} = 1$. Therefore, the link between $x_{i,j}$ and its path variables can be encoded using the following SMT constraint:*

$$\texttt{Equals}(x_{i,j}, x_{i,j-1} \vee x_{i,j,0} \vee \cdots \vee x_{i,j,\lambda-1}).$$

Given a system of connection relations $(X, \mathcal{R})$, a subset of target variables $\mathcal{T} \subset X$, and a positive integer β as the depth of knowledge propagation, we construct an SMT model which decides whether a guess basis of size up to α for $\mathcal{T}$ exists. To do so, we firstly generate the state and path variables corresponding to β steps of knowledge propagation. Assuming that $\mathcal{X} = \{x_{i,j} : 0 \leq j \leq n-1, 0 \leq i \leq \beta\}$, after adding the SMT constraints linking the state and path variables to model the knowledge propagation according to Proposition 11 and 12, we add the two following conditions to the SMT model:

1. For all $0 \leq i \leq n-1$: if $x_i \in \mathcal{T}$ then $\texttt{Equals}(x_{i,\beta}, 1)$,
2. $\sum_{i=0}^{n-1} x_{i,0} \leq \alpha$, which can be described as follows using $e = \lceil \log_2(n) \rceil$:

$$\texttt{BVULE}(\texttt{BVAdd}(\texttt{BVZExt}(x_{i,0}, e), \ldots, \texttt{BVZExt}(x_{i,n-1}, e)), \alpha), \tag{1}$$

The first condition describes that all variables of $\mathcal{T}$ must be known at the end of knowledge propagation and the second one imposes an upper bound of α on the size of the guess basis. The resulting SMT model yields as its solution a guess basis of size up to α for $\mathcal{T}$, or returns UNSAT if none exists.

We can also use this model to solve the dependency problem. For example, assume that we want to know whether the variable y is independent of $x_1, \ldots, x_n \subset X$. It is sufficient to consider $X \setminus \{x_1, \ldots, x_n\}$ as the known variables and y as the target variable, and then check if there exists a guess basis of size zero. If not, y depends on $\{x_1, \ldots, x_n\}$. As an application of this problem, we can check whether an output variable of a block or stream cipher depends on a certain set of input variables. Moreover, we can find the maximum number of deduced variables when the number of guessed variables should be at most k. To do so, we replace Line 20 and Line 21 in Algorithm 1 with constraints $\sum_{i=0}^{n-1} x_{i,0} \leq k$, and $\max \sum_{i=0}^{n-1} x_{i,\beta}$, respectively.

Encoding Using SAT The Boolean satisfiability problem (SAT) is the first known NP-complete decision problem [16] to decide whether a Boolean formula in conjunctive normal form (CNF) is satisfiable. The most typical algorithms used in modern SAT solvers are based on the original idea of Conflict-Driven Clause-Learning (CDCL) [51]. Although the SAT problem is an NP-complete problem, many SAT instances including the instances we will generate in this paper can often be solved quickly in practice.

To describe the problem of knowledge propagation and deciding whether there exists a guess basis of size up to α for a given system of connection relations and a subset of target variables, we need to encode links between the state and the path variables, as well as boundary conditions into a CNF formula. Given that all SMT constraints for the links are simple propositional formulas and every propositional formula can be easily converted into an equivalent formula in CNF form, we can model the knowledge propagation as a SAT instance with ease.

When it comes to coding the problem of deciding whether a guess basis of size up to α exists, the only constraint which is not straightforward to describe in CNF form is constraint 1, i.e., $\sum_{i=0}^{n-1} x_{i,0} \leq \alpha$, since integer addition is an unnatural operation in SAT language. If all $x_{i,0}$ are binary variables, such constraints are called cardinality constraints and belong to a larger problem class called Pseudo-Boolean constraints. A naive approach to convert the cardinality constraint into CNF form is enumerating all possible combinations of no more than α out of n variables being true. In other words, one can consider the conjunction of $\binom{n}{\alpha+1}$ clauses $\bigwedge_{i_0,\ldots,i_\alpha}(\neg x_{i_0} \wedge \cdots \wedge \neg x_{i_\alpha})$, which is not feasible when n or k is large.

Encoding cardinality constraint into CNF is a well-studied topic for which a large number of methods have been presented such as sequential counters [64], cardinality networks [3], modulo totalizer [57], and iterative totalizer [52]. The most common idea to encode the cardinality constraint is to introduce new dummy variables to lower the number of constraints, since the sizes of variables and constraints both have an important effect on the performance of solving. For example, the sequential counters method is used by the authors of [50] and [65] to automatic search for linear and differential characteristics, respectively.

We encode the cardinality constraint $\sum_{i=0}^{n-1} x_i \leq \alpha$ using sequential counters. For this, we introduce new variables $u_{i,j} (0 \leq i \leq n-2, 0 \leq j \leq \alpha)$. By adding the following clauses to the CNF model, where $1 \leq i \leq n-2$, and $1 \leq j \leq \alpha - 1$, it becomes unsatisfiable when the cardinality is larger than α:

$$\begin{cases} (\neg x_0 \vee u_{0,0}), \quad (\neg x_{n-1} \vee \neg u_{n-2,\alpha-1}), \\ (\neg x_i \vee u_{i,0}), \quad (\neg u_{i-1,0} \vee u_{i,0}), \quad (\neg x_i \vee \neg u_{i-1,\alpha-1}), \\ (\neg u_{0,j}), \\ (\neg x_i \vee \neg u_{i-1,j-1} \vee u_{i,j}), \quad (\neg u_{i-1,j} \vee u_{i,j}). \end{cases}$$

4 From Guess Basis to Gröbner Basis

As discussed in the previous section, to model the propagation of information in all CP-based approaches (including MILP, SMT, SAT, and CP) for solving the guess-and-determine problem, we have to define some additional state variables to represent the different steps of knowledge propagation, as well as a critical parameter to specify the depth of knowledge propagation, i.e., β in Algorithm 1. In contrast to the CP-based approaches for solving the guess-and-determine problem in which the depth of knowledge propagation must be specified at first, Danner and Kreuzer in [21] proposed a new algebraic approach to model the guess-and-determine problem in which not only there is no need to specify the depth of search, but also it guarantees to output a guess basis of minimal length.

In this section, we briefly recall the method introduced in [21] to translate the guess-and-determine problem to the problem of computing the reduced Gröbner basis of a Boolean polynomial ideal. We extend their model to take the target variables and known variables into account, allowing us to model the key-bridging problem as well. According to [21], the system of connection relations is translated to a binomial ideal such that its reduced Gröbner basis includes at least one guess basis of minimum length. Therefore, this approach has two prominent steps including the translation of connection relations to a Boolean polynomial system and next computing the reduced Gröbner basis.

Given a system of connection relations $(X, \mathcal{R})$, where $X = \{x_0, \ldots, x_{n-1}\}$, without loss of generality we can assume that all of the relations are implication relations. For each implication relation such as $x_{i_0}, \ldots, x_{i_{m-2}} \Rightarrow x_{i_{m-1}}$, we replace each variable x_{i_k} with a Boolean variable X_{i_k} representing whether it is known. This yields the logical formula $(\neg X_{i_0} \vee \ldots \vee \neg X_{i_{m-2}} \vee X_{i_{m-1}})$. In mathematical logic, such a logical formula with at most one positive literal is called a Horn clause and the conjunction of several Horn clauses is called a Horn formula, which is a particular kind of CNF. Accordingly, a system of connection relations can be translated to a Horn formula.

Next, the Horn formula is translated to the algebraic language. Let $\mathcal{C}$ be the derived CNF with n binary variables, and let $Sat(\mathcal{C})$ be a subset of $\{0,1\}^n$ including all solutions of $\mathcal{C}$. It is well-known that $\mathcal{C}$ can be represented via a set of Boolean polynomials $\mathcal{F}$ such that $Sat(\mathcal{C}) = \mathcal{Z}(\mathcal{F})$, where $\mathcal{Z}(\mathcal{F})$ denotes the solution set of $\mathcal{F}$. To do so, every Horn clause $(\neg X_{i_0} \vee \ldots \vee \neg X_{i_{m-2}} \vee X_{i_{m-1}})$ is translated to a binomial $x_{i_0} \cdots x_{i_{m-2}} \cdot (x_{i_{m-1}} + 1)$ in the Boolean polynomial ring $\frac{\mathbb{F}_2[x_0,\ldots,x_{n-1}]}{\langle x_0^2+x_0,\ldots,x_{n-1}^2+x_{n-1}\rangle}$. Besides this naive approach to translate a CNF to the system of Boolean polynomials in which each clause is individually translated to an ANF, there is a block-wise method [41] as well where the overlap between multiple clauses is also taken into account. Consequently, every system of connection relations can be translated to a set of Boolean binomials. The following theorem represents the relation between a minimal guess basis for a system of connection relations and the reduced Gröbner basis of its corresponding algebraic representation.

Proposition 13 (Link between guess basis and Gröbner basis [21]) *Let $(X, \mathcal{R})$ be a system of connection relations where $X = \{x_0, \ldots, x_{n-1}\}$, and $\mathcal{K}, \mathcal{T} \subseteq X$ include the known and target variables respectively. Let $\mathcal{F}$ be the set of Boolean binomials in $\frac{\mathbb{F}_2[x_0,\ldots,x_{n-1}]}{\langle x_0^2+x_0,\ldots,x_{n-1}^2+x_{n-1}\rangle}$ as the algebraic representation of $(X, \mathcal{R})$. Besides, assume that σ is a degree-compatible term ordering and J is the ideal generated by $\mathcal{F} \cup \{k + 1 : \text{for all } k \in \mathcal{K}\}$. Next, compute the reduced σ-Gröbner basis of $J + \langle t \mid \text{for all } t \in \mathcal{T}\rangle$. Then every monomial $x_{i_0} \cdots x_{i_{m-1}}$ of smallest degree in this reduced Gröbner basis corresponds to a guess basis $G = \{x_{i_0}, \ldots, x_{i_{m-1}}\}$ of minimal length.*

5 Autoguess

We have developed *Autoguess*, an easy-to-use tool that implements these techniques to find guess-and-determine attacks and key bridges. It receives a text file including the system of relations, target and known variables, as well as a positive integer as the depth of knowledge propagation[1] as input, and outputs a guess basis of minimum size. Autoguess supports all encoding methods including CP, MILP, SMT, and SAT enabling the user to utilize almost all state-of-the-art CP, MILP, SMT and SAT solvers. It also supports the Gröbner encoding method described in Section 4, which has the advantage that the user does not need to specify the depth of knowledge propagation. The output of Autoguess not only represents the guessed variables but also includes the determination flow which illustrates how the target variables can be determined from the guessed variables. Additionally, Autoguess uses `graphviz` [29] to generate a directed graph visualizing the determination flow.

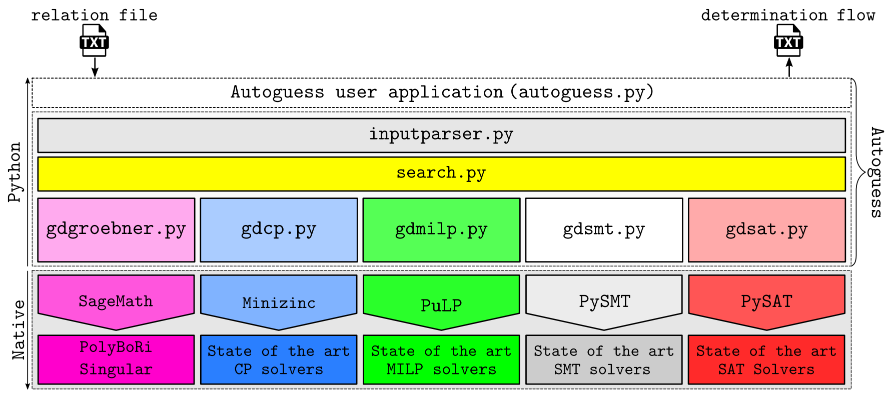


Fig. 1: The program flow of Autoguess

Figure 1 gives a high-level view of program flow in Autoguess. As illustrated in Figure 1, for the Gröbner basis approach we use SageMath [67], which provides a direct interface to a lot of state-of-the-art efficient Gröbner basis algorithms including PolyBoRi [12] and Singular [23]. Utilizing MiniZinc [54] as a CSP modeling language that is accepted by a wide range of solvers to derive the CP model in our tool, we are able to use almost all state-of-the-art CP solvers supported by MiniZinc such as Or-Tools [59][2], Gecode [34] and Choco [60]. The MILP encoding in Autoguess relies on `PuLP` [53], a high-level modeling library through which many MILP solvers such as Gurobi [37] and CPLEX [18] are available. In order to encode the guess-and-determine problem into an SMT

[1] This parameter is not required when the Gröbner algorithm is used as the solver
[2] Gold medalist of 2018, and 2019 MiniZinc Challenge in both fixed, and parallel categories, and winner of the 2020 MiniZinc Challenge in parallel category

problem, we apply PySMT [33], which allows us to use a wide range of SMT solvers supported by PySMT, including Z3 [22], CVC4 [5], Boolector [56][3], MathSAT [13] and Yices [27]. To provide a direct access to SAT solvers, we use PySAT [42] in our tool which supports many modern SAT solvers such as CaDiCaL [9], Lingeling [8], Minisat [28] and MapleSAT [32]. Additionally, PySAT supports a variety of cardinality constraint encodings including the sequential counter which was discussed earlier in Line 23. We also observed that CaDiCaL performs fastest among the mentioned SAT solvers.

Example 1. In the following system of equations, F, G, and H are some bijective functions, where u, v, w, x, y, z are 16-bit variables and $c_1, \ldots, c_5$ are constants.

$$\begin{cases} F(u \boxplus v) \oplus G(x) \oplus y \oplus (z \lll 7) &= c_1 \\ G(u \oplus w) \boxplus (y \lll 3) \boxplus z &= c_2 \\ F(w \oplus x) \boxplus y \oplus z &= c_3 \\ F(u) \oplus G(w \boxplus z) &= c_4 \\ F(u) \cdot G(w \lll 7) \boxplus H(z \oplus v) &= c_5 \end{cases} \tag{2}$$

To find a minimal guess basis for Equation (2), we introduce a variable $t = F(u) \cdot G(w \lll 7)$ to encode the connection relations corresponding to Equation (2) in a text file as follows:

```
# Comments
connection relations
u, v, x, y, z
u, w, y, z
w, x, y, z
u, w, z
u, w => t
t, z, v
end
```

Assuming that the name of the text file is `relations.txt` we call Autoguess as follows:

```
autoguess.py -i relations.txt --maxsteps 5 --solver cp
```

As the output, Autoguess generates a text file including the guess basis and its correspoding determination flow, as well as a graph visualizing the determination flow, as in Figure 2.

The user can choose the encoding method as well as the solver. For example, to find a guess basis via the Gröbner basis-based method, we can use the following command:

```
autoguess.py -i relations.txt --solver groebner
```

Besides, we can specify the target and known variables in the input text file as follows:

[3] Winner of the SMT-COMP 2019 in the bit-vector track [39]

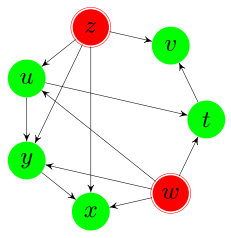


Fig. 2: Determination flow of variables in Equation (2) when w, z are 🔴 guessed variables, the others are 🟢 deduced variables.

```
# Comment
connection relations
u, v, x, y, z
u, w, y, z
w, x, y, z
u, w, z
u, w => t
t, z, v
target
u
v
x
known
w
end
```

5.1 Preprocessing Phase

When translating a system of equations to a system of connection relations, we only consider the connectivity relations between the variables and not the algebraic structure of the original system of equations. This is why neither the CP-based nor the Gröbner basis-based method can exploit the algebraic structure of the given system of equations. On the other hand, some cryptographic primitives can be simply described via algebraic equations over the same algebraic structure. The following example shows that by taking the algebraic structure into account, we might be able to achieve a better result.

Example 2. Consider the following system of equations over $\mathbb{F}_2$.

$$F = \begin{cases} x_0x_2 + x_1x_3 + x_0 + x_2 + x_3 = 0 \\ x_1x_2 + x_0x_3 + x_2x_3 + x_0 + x_1 + x_3 = 0 \\ x_1x_3 + x_2x_3 + x_0 + x_2 + 1 = 0 \\ x_0x_1 + x_0x_2 + x_1x_2 + x_2 + x_3 + 1 = 0 \\ x_0x_1 + x_1x_2 + x_0x_3 + x_2 = 0 \\ x_0x_2 + x_0x_3 + x_2x_3 + x_0 + x_1 + x_2 + x_3 = 0 \\ x_0x_1 + x_1x_3 + x_0 + 1 = 0 \end{cases} \quad (3)$$

Using the CP-based or Gröbner basis-based methods, the smallest guess basis that can be found for F has 3 elements. For example, it can be seen that $\{x_0, x_1, x_2\}$ is a guess basis. Applying Gaussian elimination yields the following system of 7 equations:

$$F_2 = \begin{cases} x_3x_1 + x_1 = 0 & x_1x_0 + x_1 + x_0 + 1 = 0 \\ x_3x_2 + x_1 + x_2 + x_0 + 1 = 0 & x_2x_0 + x_2 = 0 \\ x_3x_0 + x_1 + x_2 + x_0 + 1 = 0 & x_3 + x_1 + x_0 = 0 \\ x_1x_2 = 0 \end{cases} \tag{4}$$

Here, new equations with lower weight have been derived. Taking into account that x_1x_2 is known, if we translate $F \cup F_2$ into a system of connection relations, we can find a guess basis with one variable less than before, such as $G = \{x_0, x_3\}$. The determination flow of deducing all variables from $\{x_0, x_3\}$ is represented by Figure 3.

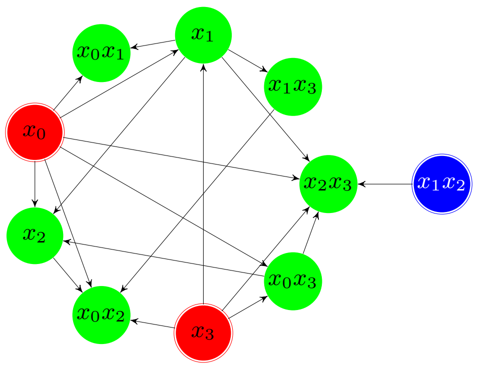


Fig. 3: Determination flow of deducing all variables from $\{x_0, x_3\}$ using Equation (3) and (4). 🔵 known variables, 🔴 guessed variables, 🟢 deduced variables.

Following the algebraic approaches to solve multivariate polynomial equations over finite field such as the Gröbner basis and XL algorithms [17, 31, 44], we can even derive further equations. To do so, we use reduced row echelon form of the degree-D Macaulay matrix which is defined as follows.

Definition 6 *[44] For any integer k, let T_k be the set of monomials of degree smaller than or equal to k, in $\mathbb{F}_2[x_0, \ldots, x_{n-1}]$. The degree-$D$ Macaulay matrix of a system of equation F, denoted by $Mac_D(F)$, is the matrix with coefficients in $\mathbb{F}_2$ whose columns are indexed by T_D and rows are indexed by the set $\{(u, f_i) \mid i \in [1; m], u \in T_{D-\deg(f_i)}\}$, and whose coefficients are those of the products $u\,f_i$ in the basis T_D.*

Although reduced row echelon form of the Macaulay matrix can be used to compute the Gröbner basis if D is large enough [48], we use relatively small D to only derive further relations. For example, the reduced echelon form of degree-3 Macaulay matrix of Equation (3) with respect to graded lexicographic order (grevlex or deglex) results in:

$$F_3 = \begin{cases} x_3x_1x_2 = 0 & x_3x_2 + x_0 = 0 & x_2x_0 + x_0 = 0 \\ x_3x_1x_0 = 0 & x_3x_0 + x_0 = 0 & x_3 + 1 = 0 \\ x_3x_2x_0 + x_0 = 0 & x_1x_2 = 0 & x_1 + x_0 + 1 = 0 \\ x_1x_2x_0 = 0 & x_1x_0 = 0 & x_2 + x_0 = 0 \\ x_3x_1 + x_0 + 1 = 0 & & \end{cases} \tag{5}$$

It can be seen that $G = \{x_0\}$ is a guess basis of $F \cup F_3$. As demonstrated in this example, using the new algebraic equations derived from the original equations using the reduced Macaulay matrix can result in a smaller guess basis. Accordingly, we are motivated to use the degree-D Macaulay matrix where D is relatively small ($D \leq 3$) as a preprocessing phase when there are some algebraic equations over finite field $\mathbb{F}_2$ among the given original equations. To take advantage of the algebraic structure of the original equations, we have included the Macaulay matrix preprocessing phase into Autoguess. Thanks to this feature, the user is able to include the algebraic equations into the input text file making a hybrid relation file consisting of connection relations as well as algebraic relations. According to the given degree for the Macaulay matrix, which is specified by the user, Autoguess applies the preprocessing phase on the algebraic equations and converts the derived equations into connection relations before encoding the guess-and-determine attack.

5.2 Early-Abort Technique

In this section we show that beside the size of the guess basis, other properties of the guess basis which can be detected by investigating the determination flow also have a significant impact on the computational complexity of the resulting GD attack.

Example 3. Consider the following system of equations where each variable represents a 16-bit word and S, L are bijective functions:

$$F_3 = \begin{cases} (x_0 \lll 3) \oplus S(x_1) \oplus x_2 = 0 & x_2 \boxplus L(x_3) = 0 \\ S(x_0) \oplus S(x_2) \oplus (x_5 \lll 2) = 0 & (S(x_3 \cdot S(x_1 \cdot x_4)) \oplus L(x_0)) \boxplus x_6 = 0 \\ L(x_1 \boxplus x_2) \boxplus x_5 = 0 & (S(x_4) \oplus L(x_6)) \boxplus x_3 = 0 \end{cases} \tag{6}$$

Let $t = S(x_3 \cdot S(x_1 \cdot x_4))$, then Figure 4 shows the output of Autoguess to find a guess basis for Equation (6) and demonstrates that all variables can be uniquely deduced from $G = \{x_0, x_4, x_5\}$. The dashed lines represent that x_1 can be deduced from $\{x_5, x_2\}$ besides $\{x_0, x_2\}$. As illustrated in Figure 4, before guessing x_4, we can uniquely determine the value of x_1 from two different equations, i.e., $(x_0 \lll 3) \oplus S(x_1) \oplus x_2 = 0$, and $L(x_0 \boxplus x_2) \boxplus x_5 = 0$. Therefore, before guessing x_4, we can use the first equation to uniquely determine the value of x_1, and reserve $L(x_0 \boxplus x_2) \boxplus x_5 = 0$ to merely check the correctness of the previous guesses. Given that the equation $L(x_0 \boxplus x_2) \boxplus x_5 = 0$ holds with probability of 2^{-16}, we can filter out a fraction 2^{-16} of wrong guesses for $(x_0, x_5) \in \mathbb{F}_2^{32}$ before guessing x_4. Thus, the computational complexity of the GD attack is reduced from 2^{48} to 2^{32}.

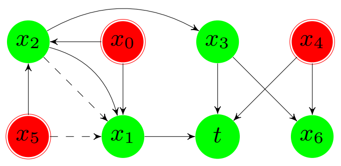


Fig. 4: Determination flow of deducing all variables from $\{x_0, x_5, x_4\}$ using Equation (6). Dashed lines represent that besides $\{x_0, x_2\}$, $\{x_2, x_5\}$ also uniquely determines the value of x_1.

Beside the variable deduction from multiple independent paths, the unused equations between deduced variables before guessing the entire basis can be used for an early abortion of wrong guesses as well. If a variable deduction from multiple independent paths or an unused relation between deduced variables appears after guessing the entire guess basis, we can still use them for checking the correctness of our guesses, but not for early abortion. As we will see in the next sections, we can use this technique to reduce the data complexity of our GD attack on block ciphers. To help the user to simply detect the unused equations and the variables that can be deduced from multiple paths, Autoguess returns all unused equations as well as those variables deducing from multiple paths, in addition to the determination flow. Given that applying the preprocessing phase may result in some redundant connection relations, the dependency between multiple paths deducing one variable should be manually checked if we aim to use both preprocessing and early abortion technique.

6 Application to Automatic Search for Key Bridges

In order to demonstrate the usefulness of our tool, we show its application to automatic search for key bridges in both bit-oriented and word-oriented key schedules.

6.1 Application to PRESENT

PRESENT is an ISO-standard ultra-lightweight SPN block cipher taking a 64-bit plaintext and 80-bit (or 128-bit) key $K = \kappa_0 \ldots \kappa_{79}$ (or $K = \kappa_0 \ldots \kappa_{127}$) as input, and returns a 64-bit ciphertext. The key schedule of PRESENT which include the nonlinear operation S-box, are described in Algorithm 2 and 6 for key length of 80 and 128 respectively.

Algorithm 2: Key schedule of PRESENT-80

```
Input: A master key $K = \kappa_0 \cdots \kappa_{79}$ of 80 bits, a number of rounds $r$ where $r \leq 31$
Output: $r + 1$ round sub-keys $K_i$ of 64 bits
1 $K_0 \leftarrow \kappa_0 \cdots \kappa_{63}$;                ▷ Extract first round sub-key;
2 for $i = 1 \rightarrow r$ do
3     $\kappa_0 \ldots \kappa_{79} \leftarrow \kappa_{61} \ldots \kappa_{79}\kappa_0 \ldots \kappa_{60}$;     ▷ Rotate 19 bits to the right;
4     $\kappa_0\kappa_1\kappa_2\kappa_3 \leftarrow S(\kappa_0\kappa_1\kappa_2\kappa_3)$;     ▷ Apply S-box on leftmost nibble;
5     $\kappa_{60}\kappa_{61}\kappa_{62}\kappa_{63}\kappa_{64} \leftarrow \kappa_{60}\kappa_{61}\kappa_{62}\kappa_{63}\kappa_{64} \oplus i$;     ▷ Add round counter;
6     $K_i \leftarrow \kappa_0 \ldots \kappa_{63}$;                ▷ Extract round sub-key;
7 return $\{K_i\}_{i=0}^{r}$;
```

Given that round counter i in line 6 of Algorithm 2 is known at each round, the round counter addition does not have any impact on knowledge propagation. Hence, to construct the system of connection relations corresponding to the key schedule of PRESENT-80, we only consider rotation and S-box operations. To model the S-box we assume that the output bits are deduced if all input bits are known, and vice versa. Hence we use $2n$ implication relations to model each S-box.

Let $k_{0,r}, \ldots, k_{79,r}$ represent whether the key bits $\kappa_0, \ldots, \kappa_{79}$ are known in round r, where $0 \leq r \leq 31$, and $k_{0,0}, \ldots, k_{79,0}$ correspond to the master key bits. Since round counter are known, the system of connection relations for R rounds of key schedule is

$$\begin{gathered} k_{r+1,i} + k_{r,(i+61 \mod 80)}, \\ k_{r,0}, k_{r,1}, k_{r,2}, k_{r,3} \Rightarrow k_{r+1,i}; \quad k_{r+1,0}, k_{r+1,1}, k_{r+1,2}, k_{r+1,3} \Rightarrow k_{r,i} \text{ for } 0 \leq i \leq 3, \end{gathered} \tag{7}$$

where $0 \leq r \leq R-1$. Similarly, to model R rounds of key schedule of PRESENT-128, it is sufficient to use the following relations alongside the relations in Equation (7), where $0 \leq r \leq R-1$:

$$k_{r,4}, k_{r,5}, k_{r,6}, k_{r,7} \Rightarrow k_{r+1,i}; k_{r+1,4}, k_{r+1,5}, k_{r+1,6}, k_{r+1,7} \Rightarrow k_{r,i} \text{ for } 4 \leq i \leq 7. \tag{8}$$

The best attacks on PRESENT so far are the linear attacks on 28 rounds of both variants of this cipher, which have been proposed at EUROCRYPT 2020 [38]. The authors of [38] introduced a new generalized and efficient key recovery technique for linear cryptanalysis and used their method to improve the linear attacks on 26 and 27 rounds of both variants of PRESENT, and successfully applied it to provide the first attack on 28 rounds of both variants of PRESENT. They also tried to use the dependency relationships between the sub-key bits involved in the key recovery to reduce the time complexity of their general key recovery algorithm in simple, multiple and multidimensional linear cryptanalysis. Although it is possible to use the dependency relationship between the sub-key bits in their key recovery algorithm, they admit that have been unable to provide an efficient general algorithm which takes account of all dependency relationships between the key-bits involved in the key recovery.

In total, 96 of the key bits need to be guessed in the 26-round key recovery attack on PRESENT-80 in [38]:

$$\begin{aligned}T =&\{k_{0,16\sim 47}, k_{1,20\sim 27}, k_{1,36\sim 43}, k_{25,0}, k_{25,2}, k_{25,8}, k_{25,10}, k_{25,16}, k_{25,18}, k_{25,24},\\ &k_{25,26}, k_{25,32}, k_{25,34}, k_{25,40}, k_{25,42}, k_{25,48}, k_{25,50}, k_{25,56}, k_{25,58}\}\cup\\ &\{k_{26,2\cdot i} :\ 0 \leq i \leq 31\}.\end{aligned}$$

Exploiting the dependency relationships between the involved key bits, the authors of [38] showed that they can all be deduced from 61 bits. However, using Autoguess running on a single core of Intel Core i9 processor at 3.6 GHz, we can find the minimal guess basis K_T of size 60 in less than 3 seconds with the Gröbner basis approach, which includes the following variables:

$$\{k_{26,2\cdot i} : 0 \leq i \leq 7\}\cup\{k_{6,42}, k_{26,15\sim 22}, k_{26,24}, k_{26,26\sim 63}, k_{26,67}, k_{26,69}, k_{26,75}, k_{26,77}\}.$$

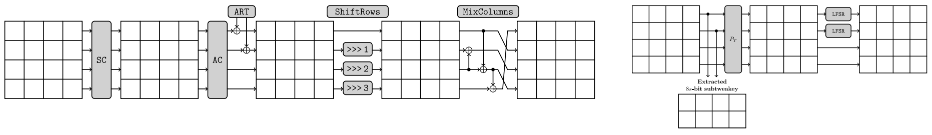
According to [38], the cost of computing the multiple linear cryptanalysis statistic throughout the analysis phase of the multiple linear attack on 26 rounds of PRESENT is $M_2 \cdot 2^{|K_T|}$, where $M_2 = 16$. Consequently, our finding reduces the time complexity of the analysis phase from 2^{65} in [38] to 2^{64}. However, the total complexity of the attack remains the same as [38], because the bottleneck of the attack is the search phase with the time complexity of $2^{68.2}$.

Moreover, to compute the time complexity of the analysis phase in the 28-round multiple linear attack on PRESENT-128 [38], it is supposed that all the involved key bits can be deduced from 114 bits, whereas the minimum-size guess basis for the involved key bits that we could discover using our tool would include 115 bits. Contacting the authors of [38] concerning this observation, they confirmed that it is a typo, and they have also discovered a guess basis of size 115 throughout their analysis. Consequently, the time complexity of the analysis phase in the multiple linear attack on 28 rounds of PRESENT-128 in [38] is expected to be more than $2^{121.58}$, whereas the authors of [38] claimed that it should be less than 2^{121}. However, the total time complexity of 28-round attack on PRESENT-128 remains at 2^{122}, as the analysis phase is not the bottleneck of the attack.

6.2 Application to SKINNY

SKINNY [6] is a family of lightweight tweakable block ciphers with several block and tweakey sizes, where the tweakey state can contain both key and tweak material. Each instance of the SKINNY family is represented by SKINNY-n-t, where n and t denote the block size and tweakey size, respectively ($n \in \{64, 128\}, t \in \{n, 2n, 3n\}$). The internal state of SKINNY can be viewed as a 4×4 array of nibbles (for 64-bit block size) or bytes (for 128-bit block size), in which entries are arranged row-wise. As illustrated in Figure 5a, five basic operations are performed on the internal state in each round of SKINNY. The tweakey state is also viewed as a collection of z 4×4 arrays, where $z = t/n$. We denote these arrays by $TK1$ when $z = 1$, $TK1$ and $TK2$ when $z = 2$, and by $TK1$, $TK2$ and

$TK3$ when $z = 3$. The tweakey arrays are updated according to Figure 5b, which illustrates one round of the tweakey schedule. It should be noted that SKINNY's tweakey schedule is linear and thus more amenable to different techniques for finding minimal guess bases based on linear algebra in some cases. However, our approach has the advantage that it can be directly merged into existing CP and optimization models and thus take the key schedule into account when searching for the best attacks, rather than only finding the best attacks based on a given distinguisher.



(a) Round function **(b)** Tweakey schedule

Fig. 5: The SKINNY round function and key schedule

Let $TKi[j]$ denotes the jth cell of TKi where $i \in \{1, 2, 3\}$. SKINNY does not use a whitening tweakey. In each round of its tweakey schedule, firstly, a permutation P_T is performed on the cell positions of all tweakey arrays, i.e., $TKi[j] \leftarrow TKi[P_T[j]]$, for all $i \in \{1, 2, 3\}, 0 \leq j \leq 15$, where $P_T = [9, 15, 8, 13, 10, 14, 12, 11, 0, 1, 2, 3, 4, 5, 6, 7]$. Then, for the SKINNY versions where $TK2$ and $TK3$ are used, every cell of the first and second rows of $TK2$ and $TK3$ are individually updated with an LFSR. Finally, the first and second rows of all tweakey arrays are extracted and bitwise XORed to the cipher internal state. One can refer to [6] for detailed description of the SKINNY cipher.

For SKINNY versions where $TK1$, and $TK2$ are used, assume that $TK1_r[j]$ and $TK2_r[j]$ denote the jth cell of tweakey arrays $TK1$ and $TK2$ at round r respectively, where $0 \leq r \leq 35, 0 \leq j \leq 15$, and $TK1_0 = TK1$, and $TK2_0 = TK2$. Besides, let $TK_r[j]$ represents the jth cell of sub-tweakey which is XORed to the cipher internal state at round r, where $0 \leq j \leq 7$. Accordingly, $TK_r[j] = TK1_r[j] \oplus TK_r[j]$, for all $0 \leq j \leq 7$, and $0 \leq r \leq 48$. Given that $TK_r[j] = TK1_r[j] \oplus TK_r[j]$ is a symmetric relation, and one can determine the value of $TKi_r[j]$ by knowing the value of $TKi[j]$ for all $i \in \{1, 2\}$, and $0 \leq j \leq 15$, the following symmetric connection relations hold:

$$\left[TK_r[j], TK1[P_T^{(r)}[j]], TK2[P_T^{(r)}[j]]\right], \text{ for } 0 \leq j \leq 7,\ 0 \leq r \leq 48, \tag{9}$$

where $P_T^{(r)}$ represents the rth iteration of P_T. For SKINNY versions where $TK3$ is used, assuming that $TK3_r[j]$ represents the jth cell of tweakey array $TK3$ at round r where $TK3_0 = TK3$, we have $TK_r[j] = TK1_r[j] \oplus TK2_r[j] \oplus TK3_r[j]$. Hence, similarly the following symmetric connection relations will hold:

$$\left[TK_r[j], TK1[P_T^{(r)}[j]], TK2[P_T^{(r)}[j]], TK3[PT^{(r)}[j]]\right], \text{ for } 0 \leq j \leq 7,\ 0 \leq r \leq 56. \tag{10}$$

At FSE 2019, Ankele *et al.* [2] proposed a related-tweak zero-correlation attack on 20 rounds of `SKINNY-64-128` and 23 rounds of `SKINNY-64-192`. In the following, we show how to improve their attacks.

Related-Tweakey ZC Attack on 20 Rounds of `SKINNY-64-128` In the 20-round related-tweak zero-correlation attack on `SKINNY-64-128` proposed in [2], in total, 20 sub-tweakey nibbles are guessed in the key-recovery attack, indicated by crosses in Figure 6. Thus, the size of the involved secret sub-tweakey is $20 \times 4 = 80$ bits.

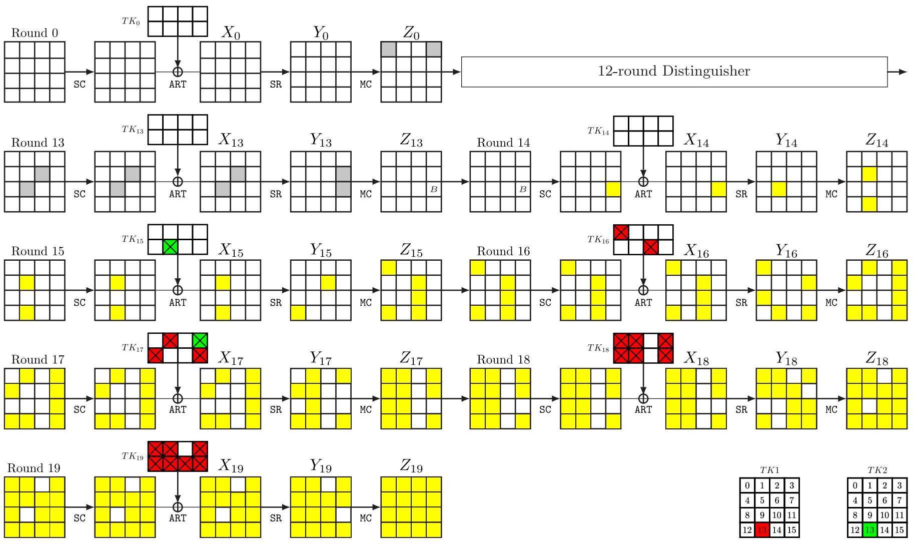


Fig. 6: Related-tweak zero-correlation key-recovery attack on 20-round `SKINNY-64-128` [2]. In the original attack, 20 nibbles must be guessed (marked by $\times$). However, they can be deduced from 19 sub-tweakey nibbles (dark red). From these, all green nibbles can be deduced.

Using our tool, we prove that these 20 sub-tweakey nibbles can be deduced by guessing 19 sub-tweakey nibbles (dark red). As illustrated in Figure 6, the following sub-tweakey nibbles (marked by $\times$) are involved in the key recovery:

$$T = \{TK_{15}[5], TK_{16}[0], TK_{16}[6], TK_{17}[1], TK_{17}[3], TK_{17}[4], TK_{17}[7], TK_{18}[0], TK_{18}[1], TK_{18}[3 \sim 5], TK_{18}[7], TK_{19}[0], TK_{19}[1], TK_{19}[3 \sim 5], TK_{19}[6], TK_{19}[7]\}.$$

Considering the connection relations in Equation (9), and T as the set of target variables in our tool, we found that the following set of variables $\mathbb{G}$ (dark red in Figure 6) is sufficient:

$$\mathbb{G} = \{TK1[13], TK_{16}[0], TK_{16}[6], TK_{17}[1], TK_{17}[4], TK_{17}[7], TK_{18}[0], TK_{18}[1], TK_{18}[3], TK_{18}[4], TK_{18}[5], TK_{18}[7], TK_{19}[0], TK_{19}[1 \sim 4], TK_{19}[5], TK_{19}[6], TK_{19}[7]\}.$$

The following determination flow as an output of our tool proves that all sub-tweakey nibbles highlighted in green can be deduced by knowing the variables of $\mathbb{G}$:

$$TK1[13], TK_{19}[7] \Rightarrow TK2[13],\ TK1[13], TK2[13] \Rightarrow TK_{17}[3],\ TK1[13], TK2[13] \Rightarrow TK_{15}[5].$$

Therefore, the actual number of guessed sub-tweakey nibbles is 19 nibbles with the key-bridging technique. Referring to Table 3 in [2], by guessing $TK1[13]$ (instead of $TK_{17}[3]$) in step 8, we can determine the value of $TK_{15}[5]$ in the final step. Therefore, the time complexity can be computed as:

$$3 \times 2^{64} + 2^{72} + 2^{76} + 2 \times 2^{80} + 2^{84} + 2 \times 2^{92} + 3 \times 2^{88} \approx 2^{93.13}.$$

Given that one structure filters incorrect guesses by a factor of 2^{-4} and in total 76 key bits should be guessed, $76/4 = 19$ structures are sufficient to uniquely determine the secret key. Consequently, the total time complexity is $2^{97.36}$, the data complexity is $19 \times 2^{64} \approx 2^{68.25}$, and the memory complexity is 2^{82} 64-bit blocks.

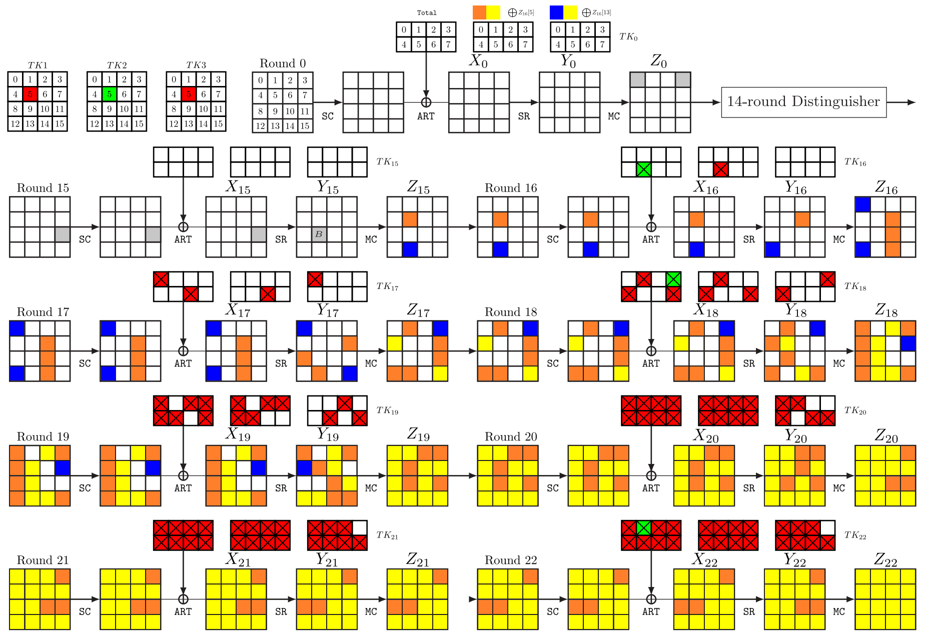


Fig. 7: Key recovery in related-tweak zero-correlation attack on 23-round SKINNY-64-192 [2].

Related-Tweakey ZC Attack on 23 Rounds of SKINNY-64-192 In the 23-round related tweak zero-correlation attack on SKINNY-64-192 proposed in [2], 37 secret sub-tweakey nibbles are involved which are denoted by a cross in Figure 7. Using the meet-in-the middle technique for integral attacks, the authors of [2] evaluate $\bigoplus Z_{15}[5]$ and $\bigoplus Z_{15}[13]$ independently to see whether the involved sub-tweakey nibbles satisfy $\bigoplus Z_{15}[5] = \bigoplus Z_{15}[13]$. Given that for a random permutation $\bigoplus Z_{15}[5] = \bigoplus Z_{15}[13]$ holds with probability of 2^{-4}, and assuming that the involved sub-tweakey nibbles take $2^{(37\times4)} = 2^{148}$ possible values, the authors of [2] claim that $148/4 = 37$ different structures are needed to uniquely determine the involved secret key. However using our tool, we prove that 37 involved sub-tweakey nibbles can be deduced from 36 sub-tweakey nibbles. Hence, the involved sub-tweakey nibbles take $2^{36\times4} = 2^{144}$ values, and 36 different structures will be enough to uniquely determine the secret key.

According to the authors' claim in [2], to compute $\bigoplus Z_{15}[5]$, a set of 34 sub-tweakeys corresponding to the cells labeled orange and yellow in Figure 7 are involved, which is represented by T_1. On the other hand, a set of 25 sub-tweakey nibbles corresponding to the cells highlighted in blue and yellow in Figure 7, are involved in evaluation of $\bigoplus Z_{15}[13]$ which is denoted by T_2. Although there is no guess basis of size less than 34 and 25 for T_1 and T_2 respectively, we discovered that there is a guess basis of size 36 for $T_1 \cup T_2$. Let the following set of variables $\mathbb{G}$ be guessed:

$$\begin{aligned} G = \{ & TK1[5], TK3[5], TK_{17}[0,6], TK_{18}[1,4,7], TK_{19}[0,2,3,4,5,7], TK_{20}[0\sim7], \\ & TK_{21}[0\sim7], TK_{22}[0], TK_{22}[2\sim7]\}. \end{aligned}$$

Considering the connection relations of Equation (10), and set $T_1 \cup T_2$ as the target set, $TK2[5]$, $TK_{16}[5]$, $TK_{18}[3]$ and $TK_{22}[1]$ can be deduced as follows:

$$\begin{aligned} &TK1[5], TK3[5], TK_{20}[7] \Rightarrow TK2[5], && TK1[5], TK2[5], TK3[5] \Rightarrow TK_{16}[5], \\ &TK1[5], TK2[5], TK3[5] \Rightarrow TK_{18}[3], && TK1[5], TK2[5], TK3[5] \Rightarrow TK_{22}[1]. \end{aligned}$$

Hence, G is a guess basis for $T_1 \cup T_2$, and all 37 involved sub-tweakey nibbles take $2^{(36\times4)} = 2^{144}$ values, and 36 structures are enough to uniquely determine the secret key. In conclusion, the total time complexity is $36\times(2^{150.4}+2^{112.3}) \approx 2^{155.57}$, the data complexity is $36 \times 2^{68} \approx 2^{73.17}$, and the memory complexity is 2^{138} 64-bit blocks.

6.3 Application to `LBlock`

As another application of our tool on a cipher with nonlinear key schedule, we apply it on `LBlock` [69]. Algorithm 7 describes the detailed key schedule of `LBlock`, where S_8 and S_9 are two 4-bit S-boxes defined in [69], where the full description of cipher can be found in as well. In the following examples, we denote the input block in round i by $X^i = X^i_L || X^i_R$, the concatenation of two 32-bit words, and the 32 sub-keys generated by the key schedule algorithm are denoted by K_i for $0 \le i \le 31$.

Application to Integral Attack on 24 Rounds of `LBlock` The best known single-key attack on `LBlock` in terms of the number of attacked rounds so far, except for biclique attack [68], is the 24-round integral attack proposed by Cui *et al.* [19]. The authors exploit the relations between the sub-keys to reduce the time complexity of the attack. Here, we show that our tool can automatically detect the relations between the key bits and produce the determination flow through which the involved key bits are deduced from the guessed variables. Let $X^i[j]\{k\}$ represent the kth bit of $X^i[j]$, where $X^i[j]$ denotes the jth nibble of X^i, for $0 \leq i \leq 31$ and $0 \leq j \leq 3$. To mount a key-recovery attack on 24 rounds of `LBlock`, the authors of [19] use a 17-round integral distinguisher based on which the correctness of $\bigoplus Z^{17}[4]\{3,2\} = \bigoplus X_L^{18}[4]\{3,2\}$ must be checked. Thanks to the meet-in-the-middle technique, the authors of [19] compute $\bigoplus Z^{17}[4]$ and $\bigoplus X_L^{18}$ independently to further reduce the time complexity of the attack. Let $k_{r,i}$, denote the ith bit of the sub-key in round r, where $k_{r,0}$ is the least significant bit of k_r, for $0 \leq i, r \leq 31$. As the key schedule of `LBlock` is similar to the key schedule of `PRESENT`, we use the same method as before to extract the connection relations for the key schedule of `LBlock`. To calculate the $\bigoplus Z^{17}[4]$, 80 key bits are involved:

$$T_1 = \{k_{17,8\sim11}, k_{18,4\sim7}, k_{19,24\sim27}, k_{20,16\sim23}, k_{21,0\sim3}, k_{21,8\sim11}, k_{21,28\sim31}, k_{22,28\sim31}, k_{22,0\sim7}, k_{22,16\sim23}, k_{23,0\sim19}, k_{23,24\sim31}\}.$$

Our tool automatically detects in less than a second that all key bits in T_1 can be determined from 55 key bits:

$$G_1 = \{k_{24,50\sim79}, k_{24,38\sim47}, k_{24,21\sim29}, k_{24,0\sim3}, k_{23,33}.k_{19,33}\}$$

On the other hand, 48 key bits are involved in calculation of $\bigoplus X_L^{18}[4]$ as follows:

$$T_2 = \{k_{19,12\sim15}, k_{20,12\sim15}, k_{21,24\sim27}, k_{21,12\sim15}, k_{22,24\sim27}, k_{22,20\sim23}, k_{22,12\sim15}, k_{23,28\sim31}, k_{23,24\sim27}, k_{23,20\sim23}, k_{23,12\sim15}, k_{23,8\sim11}\}.$$

Our tool automatically detects that all 48 key bits in T_2 can be determined from the following 47 key bits:

$$G_2 = \{k_{24,71\sim79}, k_{24,59\sim66}, k_{24,42\sim49}, k_{24,34\sim37}, k_{24,30}, k_{24,17\sim20}, k_{24,8}, k_{23,34\sim36}, k_{23,29\sim31}, k_{22,34\sim36}, k_{21,34\sim36}\}$$

Hence, guessing 47 bits is sufficient to compute $\bigoplus X_L^{18}[4]$. Lastly, we applied our tool to find a guess basis for $T = T_1 \cup T_2$, and it automatically detected that all 128 involved key bits in T can be deduced from the following 69 variables:

$$G = \{k_{24,0\sim8}, k_{24,17\sim30}, k_{24,34\sim79}\}.$$

Application to Impossible Differential Attack on 23 Rounds of `LBlock` Applying our tool to find the key bridges among the key bits involved in the impossible differential attack on 23 rounds of `LBlock`, we could reproduce the same result as [49] in a couple of seconds thanks to the Gröbner basis-based method. As illustrated in Figure 16, 144 involved sub-key bits in the impossible differential attack on 23 rounds of `LBlock` can be deduced from 73 sub-key bits.

7 Application to Automatic Guess-and-Determine Attack on Block Ciphers

In this section, we demonstrate the usefulness of our tool to automatically find guess-and-determine attacks on block ciphers. In Appendix B and Appendix C we also discuss the application to GD attack on `CRAFT` and `SKINNY`, respectively.

7.1 Automatic GD Attack on `AES`

`AES` [20] is an iterated block cipher which supports 128-bit blocks and keys of lengths 128, 192, or 256 bits. Three different versions of `AES`, including `AES`-128, `AES`-192, and `AES`-256 iterate a round function 10, 12, and 14 times respectively to produce a 128-bit ciphertext. Figure 8 represents the round function of `AES`.

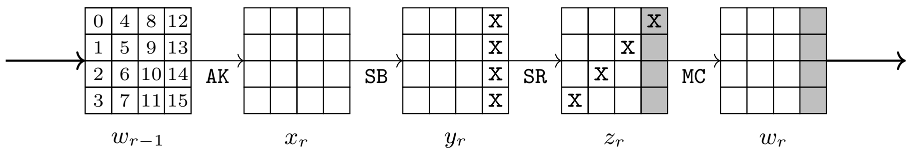


Fig. 8: Round function of `AES` [43]

As illustrated in Figure 8, the 128-bit plaintext is arranged column-wise into a 4×4 array of bytes, and then each round performs 4 basic operations on the internal state. Let $w_{r,i,j}, x_{r,i,j}$, $y_{r,i,j}$ and z_i, denote whether the jth byte in the ith row of internal state before `AK`, before `SB`, before `SR`, and before `MC` are known, respectively.

The `SB` layer applies the same 8×8 S-box on each byte of the state array. Given that $x_{r,i,j}$ is known if and only if $y_{r,i,j}$ is known for all $0 \leq i, j \leq 15$, we can assume that $x_{r,i,j} = y_{r,i,j}$. In the `SR` layer, the ith row is rotated by i bytes to the left. The `MC` layer multiplies each column of the input state by a constant 4×4 MDS matrix M. Given that M is a 4×4 MDS matrix, if $w = M \times z$, then knowing four bytes of (z, w) is sufficient to uniquely determine the remaining four bytes. Furthermore, $z_{r,i,j} = y_{r,i,(j+i) \mod 4}$ and $y_{r,i,j} = x_{r,i,j}$ for all $0 \leq i, j \leq 15$. Hence, each matrix multiplication $(w_{r,0,j}, w_{r,1,j}, w_{r,2,j}, w_{r,3,j})^t = M \times (z_{r,0,j}, z_{r,1,j}, z_{r,2,j}, z_{r,3,j})^t$ with $0 \leq j \leq 3$ in the `MC` layer can be modeled via $\binom{8}{5} = 56$ symmetric relations, each of which including five variables from the following set:

$$\{w_{r,0,j}, w_{r,1,j}, w_{r,2,j}, w_{r,3,j}, x_{r,0,j}, x_{r,1,j+1}, x_{r,2,j+2}, x_{r,3,j+3}\}.$$

Therefore, in total, $4 \times \binom{8}{5} = 224$ symmetric relations or equivalently $224 \times 5 = 1120$ implication relations are required to model the `MC` layer.

The key schedule of `AES`-128 is represented in Figure 9. Let $k_{r,i,j}$ denote whether the jth byte in the ith row of the sub-key in round r is known, where $0 \leq i, j \leq 3$, and $0 \leq r \leq 10$. Since round constant c_i is known, we can model the

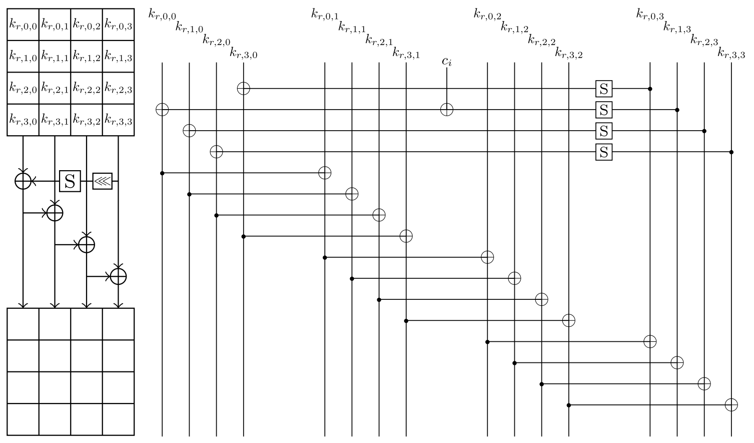


Fig. 9: AES-128 key schedule

key schedule of AES via linear algebraic relations. To do so, for each key variable $k_{r,i,3}$, $0 \leq i \leq 3$, in the third column of the key state, we define a new variable $sk_{r,i,3}$ as well as the new symmetric relation $[k_{r,i,3}, sk_{r,i,3}]$, since $k_{r,i,3}$ can be uniquely determined from $sk_{r,i,3}$ and vice versa. Therefore, we can model the key schedule as follows:

$$\begin{aligned} k_{r,i,j} \oplus k_{r+1,i,j-1} \oplus k_{r+1,i,j} &= 0, & 0 \leq i \leq 3, 1 \leq j \leq 3,\\ k_{r,3,0} \oplus sk_{r,0,3} \oplus k_{r+1,3,0} &= 0, & k_{r,1,0} \oplus sk_{r,2,3} \oplus k_{r+1,1,0} = 0,\\ k_{r,2,0} \oplus sk_{r,3,3} \oplus k_{r+1,2,0} &= 0, & k_{r,0,0} \oplus sk_{r,1,3} \oplus k_{r+1,0,0} = 0. \end{aligned}$$

The advantage of modeling the key schedule via the algebraic equations is that we can apply the preprocessing phase on the above equations to derive more relations between the key variables[4]. In the AK layer, 16 bytes of sub-key are XORed to the internal state, which can be modeled via the following connection relations:

$$[w_{r-1,i,j}, k_{r,i,j}, x_{r,i,j}] \text{ for all } 0 \leq i, j \leq 15.$$

Consider an adversary who seeks to break one full round of AES, i.e., AK ∘ MC ∘ SR ∘ SB ∘ AK, where only a single known plaintext is available. Given the system of connection relations corresponding to one round of AES as well as the known and target variables, (with or without preprocessing phase) Autoguess finds a minimal guess basis of size 6 bytes:

$$G = \{k_{0,0,1}, k_{1,0,0}, k_{1,3,1}, x_{0,2,3}, k_{0,0,2}, k_{0,0,0}\}. \tag{11}$$

It means there is a guess-and-determine attack with time complexity of 2^{48} on one full rounds of AES. Assisting the output of Autoguess, we can specify

[4] In this case we set the degree of Macaulay matrix in preprocessing phase to be 1 which is equivalent to applying Gaussian elimination on the original equations.

the number of required known plaintext/ciphertext pairs for this attack. Figure 10(right) is generated by Autoguess and visualizes the determination flow in the GD attack on one round of AES, where the known, guessed and determined variables are represented by blue, red, and green squares, respectively and dashed arrows link variables that can be deduced from multiple paths. It can be seen in Figure 10 that $k_{0,2,1}$ can be deduced from both $\{p_{2,1}, x_{0,2,1}\}$, and $\{k_{1,2,0}, k_{1,2,1}\}$. Therefore, this gives us an 8-bit condition to check the correctness of the guesses. There is a similar situation for $k_{1,3,3}$ and $k_{0,1,0}$ as well. $w_{0,0,3}$ can also be deduced either from $\{x_{1,0,3}, k_{1,0,3}\}$ or from $\{x_{0,0,3}, x_{0,3,2}, w_{0,1,3}, w_{0,2,3}\}$ (`MC` layer). Besides, according to the output of Autoguess which prints out the unused relations, we notice that two symmetric relations $[k_{0,2,0}, k_{0,3,3}, k_{1,2,0}]$, and $[k_{0,1,1}, k_{1,1,0}, k_{1,1,1}]$ have not been used during the determination which can be reserved for checking the correctness of guesses. These all give 48-bit conditions which hold with a probability of 2^{-48}. Hence, given a known pair of plaintext/ciphertext, by guessing 6 bytes we can uniquely determine the internal state as well as the secret key, without the need for an additional pair of plaintext/ciphertext for checking the correctness.

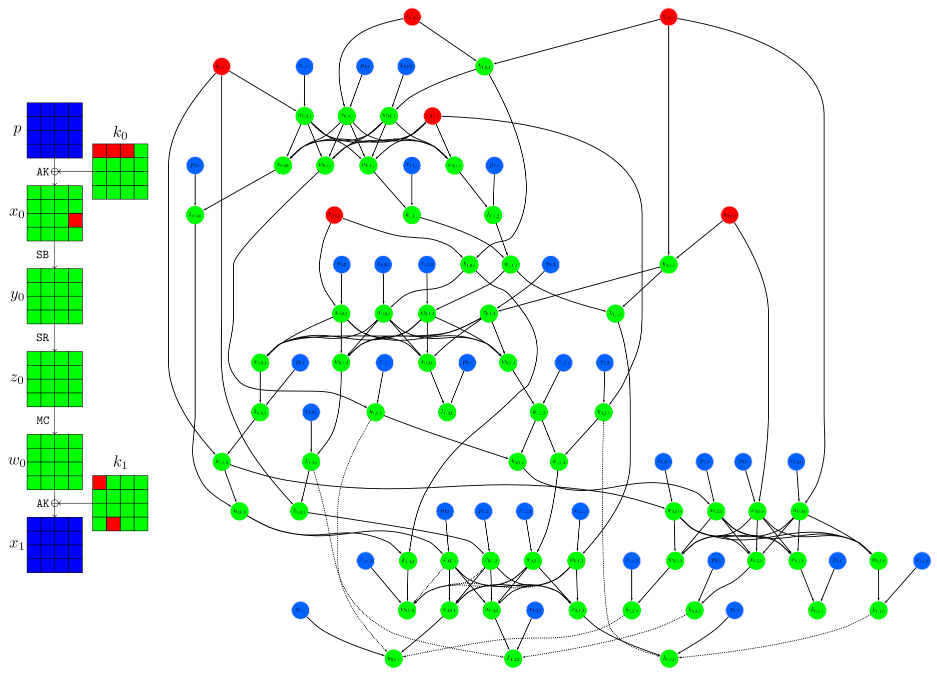


Fig. 10: Determination flow in GD attack on one full round of `AES`. Red and blue circles represent the guessed and known variables, respectively, and green circles the deduced variables.

It should be noted that this attack has been already discovered in [25], using manual approaches. However, with Autoguess running on a single-core Intel Core i9 processor at 3.6 GHz, we can discover this attack in less than 0.01 second after 14 steps of knowledge propagation when the SAT solver CaDiCal is called as the solver. Using the Gröbner basis-based method we also discovered a guess basis of size 6 bytes (after about 5 seconds on the same system).

Applying Autoguess on 2 and 3 rounds of `AES`, we find guess bases of size 11 and 15 bytes, respectively. Thanks to the output of Autoguess, in the same way as our attack on 1 round of `AES`, we make sure that only one known plaintext/ciphertext pair is sufficient to uniquely determine the unknown variables. It means that there are guess-and-determine attacks with time complexity of 2^{88} and 2^{120} on 2 and 3 rounds of `AES`, respectively, which require only one known plaintext/ciphertext pair. Figure 17 and Figure 19 represent the determination flow in guess-and-determine attacks on 2 and 3 rounds of `AES`, respectively. It is worth noting that running Autoguess on a single core Intel Core i9 processor at 3.6 GHz, and using the SAT-based method (CaDiCal), it took less than a minute, to find the GD attack on 3 rounds of `AES`, whereas it took about 10 hours when we used the Gröbner basis approach (PolyBoRi). We also used the MILP-based methods (Gurobi) to find the GD attack on `AES`, and noticed that it also much slower than the SAT-based method in this application even if we want to find only a feasible solution.

It should be mentioned that the best previous GD attacks on 1 and 2 rounds of `AES` proposed in [11] are still better by guessing one byte less than our attacks. Although the tool proposed in [11] also works on byte-level, it implements a dedicated algorithm to search for GD and MITM attacks on `AES` and hence exploits the algebraic dependencies in AES-like equations, whereas in our CP-based method we are not able to make the most of the algebraic dependencies. For instance, the 2-round GD attack on `AES` with time complexity of 2^{80} in [11] exploits the algebraic dependencies between non-consecutive sub-keys (observation 3 in [11]), which can not be completely exploited even with the preprocessing phase of Autoguess. We noticed that if we manually include the relations between non-consecutive sub-keys into the system of relations corresponding to 2 rounds of `AES`, Autoguess also finds a GD attack with a time complexity of 2^{80} on two rounds of `AES` which requires only one known plaintext/ciphertext pair as shown in Figure 18. Although the dedicated tool introduced in [11] exploits the algebraic dependencies, Autoguess gives the same result as [11] concerning the GD attack on 3 rounds of `AES`, when only one known plaintext/ciphertext pair is available.

7.2 Automatic GD Attack on `Khudra`

Khudra [47] is a lightweight block cipher designed for FPGA based platforms. It receives a 64-bit plaintext with an 80-bit master key and then iterates a generalized Feistel structure 18 times to produce a 64-bit ciphertext. After splitting the 80-bit master key into five 16-bit sub-keys $k_0, \ldots, k_4$, Khudra periodically uses two different sub-keys in each round. For more details about the specification of `Khudra`, we refer to [47]. Mehmet *et al.* [58] proposed an equivalent representation for Khudra which reveals that the effective key length for one round of Khudra is 16 bits. Thanks to this new alternative representation, they discovered a GD attack on 14 rounds of Khudra, using manual approaches. Figure 11 depicts the round function in the equivalent representation of Khudra. Here, using Autoguess we automatically rediscover the best GD attacks on 14 rounds of Khudra.

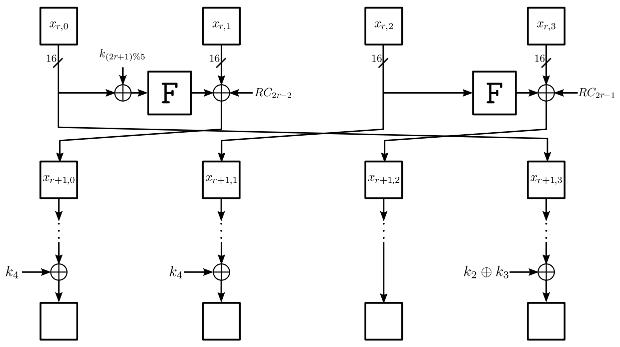


Fig. 11: The rth and last round in the equivalent representation of Khudra

Let $y = \mathtt{F}(x)$, since F is a bijective function, x is known if and only if y is known. Besides, the constants are also known. Hence, F as well as constants can be omitted in our connection relations. Assuming that $x_{r,i}$ denotes whether the ith branch at the input of round r is known, and according to Figure 11, the connection relations corresponding to R rounds of Khudra are as follows, where $0 \leq r \leq R-1$ and $d = k_2 \oplus k_3$:

$$\begin{array}{lll}
[x_{r,0}, k_{(2r+1)\%5}, x_{r,1}, x_{r+1,0}], & [x_{r,2}, x_{r,3}, x_{r+1,2}], & [x_{r,0}, x_{r+1,3}], \\
[x_{r,2}, x_{r+1,1}], & [x_{R,0}, k_4, x_{R+1,0}], & [x_{R,1}, k_4, x_{R+1,1}], \\
[x_{R,2}, x_{R+1,2}], & [k_2, k_3, d], & [x_{R,3}, d, x_{R+1,3}].
\end{array}$$

Note that $x_{0,0}||x_{0,1}||x_{0,2}||x_{0,3}$ and $x_{R+1,0}||x_{R+1,1}||x_{R+1,2}||x_{R+1,3}$ are both known, as they are corresponding to the plaintext and ciphertext, respectively. Table 3 briefly describes the time and data complexity of the discovered GD attacks by Autoguess on 1 to 14 rounds of Khudra. For 14 rounds four 16-bit variables should be guessed, which yields a GD attack with time complexity of 2^{64} in which merely 2 known plaintexts are required. Figure 20 (top) represents the determination flow in the GD attack on 14 rounds of Khudra. From the output of Autoguess, we noticed that one variable can be deduced from two independent relations, and two symmetric relations each of which inducing a 16-bit condition are not used during the determination which can be reserved for checking the correctness of guesses. Hence, there are three 16-bit conditions which can be satisfied with a probability of 2^{-48}. Given that we are required to check the correctness of 64 guessed bits, one additional known plaintext/ciphertext pair is required to uniquely determine the internal state as well as the secret key.

Table 3: Number of guessed variables in GD attack on 1 to 14 rounds of Khudra

#Rounds	1	2	3	4	5	6	7	8	9	10	11	12	13	14
#Guessed variables	0	1	1	1	2	3	3	3	3	4	4	4	4	4
#Required data (KP)	1	1	1	1	2	2	2	2	2	2	2	2	2	2

8 Application to Automatic Guess and Determine Attack on Stream Ciphers

This section is dedicated to showing the application of Autoguess for the guess-and-determine attack on stream ciphers.

8.1 Automatic GD Attack on ZUC

ZUC is a word-based stream cipher which has two versions, ZUC-128 [30] and ZUC-256 [66]. ZUC-128 takes a 128-bit key with 128-bit IV, whereas the length of key and IV in ZUC-256 is 256 and 128 bits, respectively. Although the initialization phase of ZUC-128 and ZUC-256 is different, they perform exactly the same keystream generation phase as illustrated in Figure 12.

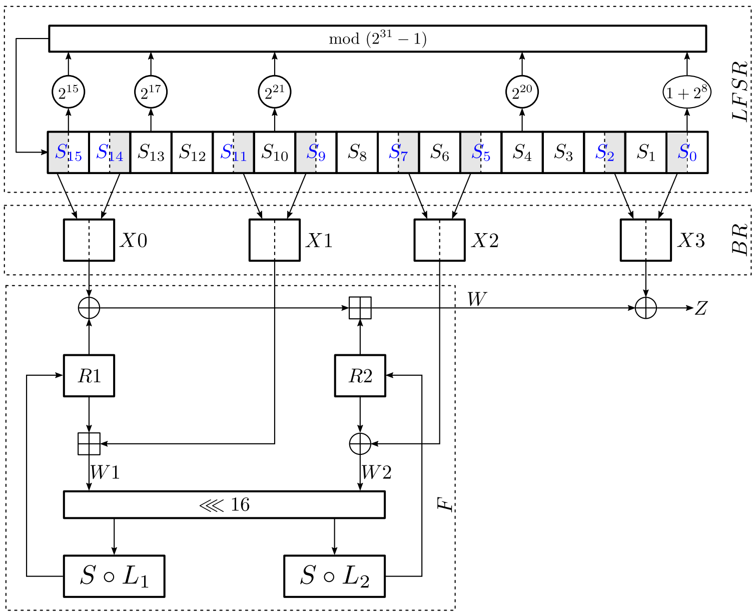


Fig. 12: The keystream generation pahse of ZUC stream cipher

The keystream generation phase of ZUC consists of three layers including a linear feedback shift register ($LFSR$); the bit-reorganization layer (BR) and a nonlinear block which is denoted by F. The $LFSR$ layer is composed of 16 cells denoted by $S_t, \dots, S_{t+15}$ at clock t, such that each cell stores 31 bits as an element from the finite field $GF(p)$, where $p = 2^{31} - 1$. The $LFSR$ part is updated according to the following relation:

$$S_{16+t} = 2^{15} S_{15+t} + 2^{17} S_{13+t} + 2^{21} S_{10+t} + 2^{20} S_{4+t} + (1 + 2^8) S_t \mod p. \quad (12)$$

Besides, if $S_{16+t} = 0$, then $S_{16+t} = p$. As the connection layer between the $LFSR$ and F, the BR layer extracts 128 bits from $LFSR$ to form four 32-bit words $X0, X1, X2$, and $X3$ such that the first three words are fed to F and the last one is XORed with the output of F to generate the keystream. If we denote the value of registers $X0, X1, X2, X3, X4$ at clock t as $X0_t, X1_t, X2_t, X3_t$, then we have $X0_t = SH_{15+t} || SL_{14+t}, X1_t = SL_{11+t} || SH_{9+t}, X2_t = SL_{7+t} || SH_{5+t}$, and $X3_t = SL_{2+t} || SH_t$, where SH_t and SL_t represent the high and low 16 bits of register S_t, respectively, i.e., $SH_t = S_t[30 \dots 15]$ and $SL_t = S_t[15 \dots 0]$. Note that, in this representation, $SL_t[15] = SH_t[0]$, i.e., the least significant bit of SH_t is the same as the most significat bit of SL_t. The nonlinear function F has two 32-bit registers $R1$ and $R2$ which takes $X0_t, X1_t, X2_t$ as the input and produces one 32-bit word which is used to generate the keystream output word Z_t as follows:

$$Z_t = ((R1_t \oplus X0_t) \boxplus_{32} R2_t) \oplus X3_t, \quad (13)$$

where $R1_t$ and $R2_t$ represent the value of registers $R1$ and $R2$ at clock t, respectively. Next, F is updated according to the following relations:

$$W1_t = R1_t \boxplus_{32} X1_t, \quad (14)$$

$$W2_t = R2_t \oplus X2_t, \quad (15)$$

$$R1_{t+1} = S\left(L_1\left(W1L_t || W2H_t\right)\right), \quad (16)$$

$$R2_{t+1} = S\left(L_2\left(W2L_t || W1H_t\right)\right), \quad (17)$$

where $W1H_t$, and $W1L_t$ represent the high and low 16 bits of $W1$ at clock t, respectively. $W2L_t$, and $W2H_t$ are defined in the same way. Here S is a 32×32 one-to-one S-box, and L_1, L_2 are two 32×32 linear functions. Because the attack in this paper uses only the keystream generation phase, we ignore the initialization phase, but refer to [30] and [66] for more details.

The best previous guess-and-determine attack on ZUC has been proposed in [36] and has a computational complexity of 2^{392} while requiring 9 keystream outputs. The GD attack in [36] is based on half-words, i.e., 16-bit blocks, by splitting the registers in the $LFSR$ and F into high and low 16 bits, where some 1-bit additional variables are introduced to link the half-words. Here, assisted by Autoguess, we propose a GD attack on ZUC with a computational complexity of 2^{390}, whereas we use the same amount of keystream outputs as [36], i.e., 9 keystream outputs.

To find the GD attack on ZUC, we use the same half-word-based model as [36], but we use Autoguess to find a guess basis rather than the manual approach. Firstly, we recall the half-word-based relations in [36]. Assume that $R1H_t$ and

$R1L_t$ represent the high and low 16 bits of $R1$ at clock t. $R2H_t$ and $R2L_t$ are defined in the same way. According to Equation (13), we have:

$$ZL_t = ((SL_{14+t} \oplus R1L_t) \boxplus_{16} R2L_t) \oplus SH_t, \tag{18}$$

$$ZH_t = ((SH_{15+t} \oplus R1H_t) \boxplus_{16} R2H_t \boxplus_{16} c1_t) \oplus SL_{2+t}, \tag{19}$$

where ZL_t, and ZH_t denote the high and low 16 bits of output word Z at clock t, and $c1_t$ represents the carry bit in modular addition $(SL_{14+t} \oplus R1L_t) \boxplus_{16} R2L_t$. Hence, we have:

$$c1_t = \begin{cases} 1 & (SL_{14+t} \oplus R1L_t) + R2L_t \geq 2^{16} \\ 0 & (SL_{14+t} \oplus R1L_t) + R2L_t < 2^{16}. \end{cases} \tag{20}$$

Splitting the Equation (14) into two 16-bit halves, we have:

$$W1L_t = R1L_t \boxplus_{16} SH_{9+t}, \tag{21}$$

$$W1H_t = R1H_t \boxplus_{16} SL_{11+t} \boxplus_{16} c2_t, \tag{22}$$

where $c2_t$ is the carry bit of modular addition $(SL_{14+t} \oplus R1L_t)$, which is defined as below:

$$c2_t = \begin{cases} 1 & R1L_t + SH_{9+t} \geq 2^{16} \\ 0 & R1L_t + SH_{9+t} < 2^{16}. \end{cases} \tag{23}$$

In the same way we can split the Equation (15) into two 16-bit parts as follows:

$$W2L_t = R2L_t \oplus SH_{5+t}, \tag{24}$$

$$W2H_t = R2H_t \oplus SL_{7+t}. \tag{25}$$

As it can be seen, all of the derived relations can be simply modeled via symmetric relations, except for Equation (20) and Equation (23) which can be modeled via the implication relations. For example, according to Equation (20) we can deduce the value of $c1_t$ if we know the values of $SL_{14+t}, R1L_t$, and $R2L_t$, i.e., $SL_{14+t}, R1L_t, R2L_t \Rightarrow c1_t$. The Equation (23) can be modeled via an implication relation in the same way. Besides the main equations, we define some trivial implication relations to link each 32-bit word to its corresponding half-words. For example, to link S_t to its corresponding half-words SH_t and SL_t, we include the following implication relations into the model:

$$S_t \Rightarrow SH_t,\ S_t \Rightarrow SL_t,\ SH_t, SL_t \Rightarrow S_t.$$

Therefore, we can generate the system of connection relations modeling the knowledge propagation through to the given number of clock cycles of `ZUC` based on half-words. In contrast to the previous system of connection relations in our paper, where all variables have the same amount of information, the system of connection relations for `ZUC` is composed of variables with different lengths. To consider the length of each variable, we use a weighted objective function in Line 21 of Algorithm 1, such that each variable is multiplied by its length.

Consequently, solving the generated model yields a guess basis of minimum weight, which corresponds to the minimum number of guessed bits to derive the internal state of ZUC.

After applying Autoguess on the system of connection relations derived from 9 clock cycles of ZUC, we noticed that finding a minimal guess basis is very time consuming such that the state-of-the-art MILP or CP solvers are not able to solve the derived optimization problem in a reasonable time. However, we are still able to find some feasible solutions which result in some guess basis smaller than the guess basis proposed in [36] by two bits. The following is one of the guess bases of size 391 bits we found using Autoguess:

$$G = \{S_5, R1_5, S_{19}, SH_{13}, S_6, S_7, S_9, S_{10}, SL_{13}[14\dots0], S_{15}, S_{16}, S_{18}, S_{20}, c2_5, SH_{12}, c1_3\}.$$

Note that it is supposed that Z_t and subsequently ZL_t, ZH_t are known for $0 \leq t \leq 8$. Thanks to the output of Autoguess, which precisely represents the determination flow, we noticed that two halves of S_{11}, i.e., SL_{11} and SH_{11} can be deduced from two independent paths. Besides, using Autoguess we could simply detect that both SL_{11} and SH_{11} are independent of $c1_3, SH_{12}$. To do so, we simply consider $G \setminus \{c1_3, SH_{12}\}$ as the known variables and $T = \{SL_{11}, SH_{11}\}$ as the target variable in the input file of Autoguess and next run Autoguess to see whether there is a guess basis of size zero. This is the case, which means SL_{11} and SH_{11} can be uniquely deduced from $G \setminus \{c1_3, SH_{12}\}$. As a consequence, we can check the condition $SH_{11}[0] = SL_{11}[15]$, before guessing $c1_3$, SH_{12} as an early abortion technique to filter out half of the wrong guesses. Algorithm 3 precisely describes our GD attack on ZUC more. Before Line 18 of Algorithm 3, 374 bits are guessed in total. Since the condition in Line 18 holds with a probability of 2^{-1}, the lines after Line 18 will be performed 2^{373} times on average, where there is a loop enumerating 2^{17} possible cases for $(SH_{12}, c1_3)$. Hence, the total computational complexity of our GD attack on ZUC is 2^{390}, in which we use 9 output keystream to determine the whole internal state.

9 Key-Recovery-Friendly Distinguishers

As we saw in the previous sections, the complexity of the attack on block ciphers is not only dependent on the distinguisher, but also depends on the actual key bits involved in the extended rounds before and after the distinguisher. However, most of the automatic CP-based methods to search for distinguishers are blind to the key recovery phase and only describe the distinguishing part. Very recently, some authors have tried to make a unified CP-model combining the distinguisher and key recovery phases to directly find a key recovery attack, but they are limited to block ciphers with linear key schedules [15, 61, 62]. To the best of our knowledge, no general CP-based model integrating the distinguishing and key recovery phases in which key-bridging can be considered for both linear and nonlinear key schedules has been introduced so far. In this section, we show

Algorithm 3: GD Attack on ZUC With Time Complexity of 2^{390}

```
   Input: Output keystream derived from 9 clock cycles of ZUC: $(Z_0, \dots, Z_8)$
1  forall $(S_5, R1_5, S_{19}, SH_{13}) \in \mathbb{F}_2^{110}$ do
2      $W1L_4, W2H_4 \Leftarrow (16)$; $R1L_4 \Leftarrow (21)$; $R2L_5 \Leftarrow (18)$; $c1_5 \Leftarrow (20)$; $c2_4 \Leftarrow (23)$;
3      forall $(S_{20}, S_7) \in \mathbb{F}_2^{62}$ do
4          $X0_5 \Leftarrow SH_{20}||SL_{19}$; $X3_5, X2_0 \Leftarrow SL_7, SH_5$; $R2_5 \Leftarrow (13)$;
5          $W1H_4, W2L4 \Leftarrow (17)$;
6          forall $(S_{16}, S_{10}, c2_5, S_9, S_{15}, S_{18}, S_6) \in \mathbb{F}_2^{178}$ do
7              $W1H_5 \Leftarrow (22)$; $W2L_5 \Leftarrow (24)$;$R2_6 \Leftarrow (17)$; $X1_4, X2_8 \Leftarrow SL_{15}||SH_{13}$;
8              $S_{21} \Leftarrow (12)$; $R1_4 \Leftarrow (14)$; $R2L_4 \Leftarrow (24)$; $R1L_6 \Leftarrow (18)$; $S_{22} \Leftarrow (12)$;
9              $X0_4 \Leftarrow SH_{19}||SL_{18}$; $SH_4 \Leftarrow (18)$; $c1_4 \Leftarrow (20)$; $R2H_4 \Leftarrow (19)$;
10             $X3_4 \Leftarrow SL_6||SH_4$; $W1L_3, W2H_3 \Leftarrow (16)$; $c1_6 \Leftarrow (20)$;
11             $X2_2, X3_7 \Leftarrow SL_9||SH_7$; $c2_6 \Leftarrow (23)$; $W1L_3, W2H_3 \Leftarrow (16)$;
12             $W1L_6 \Leftarrow (21)$; $SL_{11} \Leftarrow (25)$; $R2_4 \Leftarrow (13)$; $X0_7 \Leftarrow SH_{22}||SL_{21}$;
13             $R2H_3 \Leftarrow (25)$;
14             forall $SL_{13}[14 \dots 0] \in \mathbb{F}_2^{15}$ do
15                 $W2H_6 \Leftarrow (25)$; $R1_7 \Leftarrow (16)$; $S_3 \Leftarrow (12)$;
16                 $R2_7 \Leftarrow (13)$; $W2L_6, W1H_6 \Leftarrow (17)$;
17                 $SH_{11} \Leftarrow (24)$; $X1_0 \Leftarrow SL_{11}||SH_9$; $X1_2 \Leftarrow SL_{13}||SH_{11}$;
18                 if $SH_{11}[0] = SL_{11}[15]$ then
19                     forall $(SH_{12}, c1_3) \in \mathbb{F}_2^{17}$ do
20                         $W1H_3, W2L_3 \Leftarrow (17)$; $c2_3 \Leftarrow (23)$; $R1H_3 \Leftarrow (19)$;
21                         $R1L_3 \Leftarrow (21)$; $SL_{14} \Leftarrow (22)$; $X1_7 \Leftarrow SL_{18}||SH_{16}$;
22                         $W1_7 \Leftarrow (14)$; $W2H_7 \Leftarrow (25)$; $R1_8 \Leftarrow (16)$; $W2L_7 \Leftarrow (24)$;
23                         $R2_8 \Leftarrow (17)$; $SH_8 \Leftarrow (18)$; $R2L_3 \Leftarrow (24)$; $SL_{17} \Leftarrow (18)$;
24                         $R1H_6 \Leftarrow (22)$; $SL_8 \Leftarrow (19)$; $X2_1 \Leftarrow SL_8||SH_6$;
25                         $W1L_2, W2H_2 \Leftarrow (16)$; $W1H_2, W2L_2 \Leftarrow (17)$; $R1L_2 \Leftarrow (21)$;
26                         $R1_2 \Leftarrow (14)$; $W1L_1, W2H_1 \Leftarrow (16)$; $R2_2 \Leftarrow (15)$;
27                         $W2L_1, W1H_1 \Leftarrow (17)$; $X1_6 \Leftarrow SL_{17}||SH_{15}$; $R1_6 \Leftarrow (14)$;
28                         $W1L_5, W2H_5 \Leftarrow (16)$; $SL_{12} \Leftarrow (25)$; $X1_1 \Leftarrow SL_{12}||SH_{10}$;
29                         $R1_1 \Leftarrow (14)$; $R2_1 \Leftarrow (15)$; $W1L_0, W2H_0 \Leftarrow (16)$; $R1_0 \Leftarrow (14)$;
30                         $W2L_0, W1H_0 \Leftarrow (17)$; $R2_0 \Leftarrow (15)$; $X0_0 \Leftarrow SH_{15}||SL_{14}$;
31                         $X3_0 \Leftarrow (13)$; $SL_2 \Leftarrow X3_0$; $R2L_2 \Leftarrow (24)$; $SH_2 \Leftarrow (18)$;
32                         $SH_{14} \Leftarrow (21)$; $S_{17}(12)$; $S_1 \Leftarrow (12)$; $S_4 \Leftarrow (12)$; $S_0 \Leftarrow (12)$;
33                         if $\texttt{ZUC}^{18clks}(S_0, \dots, S_{15}, R1_0, R2_0) = (Z_0, \dots, Z_{17})$ then
34                             return $(S_0, \dots, S_{15}, R1_0, R2_0)$
35                         else
36                             Go to step 1 and try another guesses.
```

that our CP-based approach to find the key-bridges is consistent with previous CP-based methods to search for distinguishers, and hence they can be merged to directly find a key-recovery-friendly distinguisher. To do so, we extend the CP model introduced in [62] to take the key recovery as well as the key-bridging technique into account while searching for $\mathcal{DS}$-MITM distinguishers for ciphers with linear and nonlinear key schedules.

In [62], the attacks are built based on two approaches. The first approach enumerates several distinguishers where some valid $\mathcal{DS}$-MITM distinguishers are enumerated in advance by listing some solutions of the underlying CP model that only describes the distinguisher phase. Next, key recovery is mounted upon the listed distinguishers from which they pick the best one. Hence, this approach is not efficient when there are many solutions for the distinguishing phase. In the second approach, taking the key recovery attack into account, they generate a more advanced CP model to produce a key recovery attack directly. However, the authors of [62] admit that their advanced CP models are unable to completely exploit the key-bridging technique. For example, they limit the guessed sub-key variables to be in a special round which should be specified by the user in advance. Hence, by extending the CP model proposed in [62], the authors of [15] tried to propose a CP model taking the key-bridging technique into account, but only for SKINNY-128-384, which has a linear key schedule.

Our strategy to integrate the key-bridging technique into the previous CP-based frameworks for automatic search of distinguishers can be summarized as follows:

1. First, we generate the constraints describing the distinguisher part based on the previous CP-based approaches. In this step, some additional constraints can be used to limit the data, memory, and time complexity of the distinguishing phase.
2. Next, we generate the constraints modeling the active cell propagation before and after the distinguisher, based on which the involved sub-keys can be determined.
3. Then, to model the key-schedule taking the key-bridging technique into account, we generate the constraints describing the key-bridging technique based on our approach.
4. We define a few additional constraints to link the variables for the last step of knowledge propagation in the key-bridging constraints to the variables indicating the involved sub-keys in the outer rounds.
5. Finally, we look for a feasible solution minimizing the actual number of guessed sub-key variables such that all involved sub-keys can be deduced.

Note that, although the above model aims to minimize the number of actual guessed sub-keys, it can be easily modified to minimize the memory or data complexity when the number of actual guessed sub-keys is limited to be lower than or equal to some bound. Modeling the distinguisher part as well as the active cells propagation through the outer rounds (Item 1 and Item 2) has been sufficiently discussed in the previous works [61,62]. Item 3 also has been described in the previous sections. Hence, we now explain Item 4.

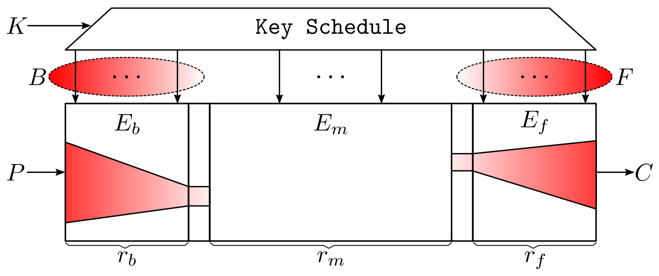


Fig. 13: A high level view of the involved key materials in a key recovery attack.

Among the variables defined in Item 2, let $ISK_{r,i}$ be a binary variable indicating whether the ith word (bit) of sub-key in round r is involved. Assuming that wk sub-key words (bits) are used in each round, as illustrated in Figure 13, suppose that $B = \{ISK_{r,i} : 0 \leq r \leq r_b - 1,\ 0 \leq i \leq wk - 1\}$, and $F = \{ISK_{r,i} : r_b + r_m \leq r \leq r_b + r_m + r_f - 1,\ 0 \leq i \leq wk - 1\}$ include the indicator variables corresponding to the involved sub-key words in the rounds before and after the distinguisher, respectively. Moreover, assuming that $\beta \in \mathbb{Z}_{\geq 0}$ denotes the depth of knowledge propagation in our key-bridging technique, let $SK_{r,i,j}$ specify whether the ith word of the sub-key in round r is known at the jth step of knowledge propagation. In contrast to our previous key-bridging models, where the target sub-keys were specified in advance, we include the following constraints to dynamically determine the target sub-key variables in our new model:

$$\{SK_{r,i,\beta} \geq ISK_{r,i} :\ 0 \leq r \leq r_b - 1 \vee r_b + r_m \leq r \leq r_b + r_m + r_f - 1,\ 0 \leq i \leq wk - 1\}.$$

Therefore, if $ISK_{r,i} = 1$ then $SK_{r,i,\beta} = 1$, which ensures that each involved sub-key variable should be deduced after β steps of knowledge propagation. Consequently, to find the minimum number of guessed sub-key variables, it is enough to minimize the number of guessed sub-key variables at the first step of knowledge propagation, i.e., $\sum_{r,i} SK_{r,i,0}$.

To model the distinguisher part as well as the active cells propagation, we use the same method as [62], and our notations in the rest of this section follow theirs.

9.1 Improved $\mathcal{DS}$-MITM Attack on `TWINE`-80

The best $\mathcal{DS}$-MITM attack on `TWINE`-80 is a 20-round attack built upon a 11-round distinguisher [62]. Thanks to combining our key-bridging technique with the automatic method to search for $\mathcal{DS}$-MITM distinguishers in [62], we discovered that using a 10-round distinguisher yields a better key recovery attack on 20-round `TWINE`-80 in terms of time complexity and memory complexity.

Complexity As illustrated in Figure 21, the $\mathcal{DS}$-MITM distinguisher requires guessing 14 nibbles (compared to 19 in [62]) and hence the time complexity of the offline phase is $2^{4\times14} \times 2^{4\times1} \times \frac{14}{20\times8} C_E \approx 2^{56.48} C_E$ (compared to $2^{76.93} C_E$ in [62]), where C_E is the time of running 20 rounds of `TWINE`-80. The memory complexity is $(2^4 - 1) \times 4 \times 2 \times 2^{4\times14} \approx 2^{62.91}$ bits (previously $2^{82.91}$). As illustrated in Figure 23, 26 sub-key nibbles are involved in the key recovery phase of our attack. With the key-bridging technique, all 26 sub-key nibbles can be deduced from 19 sub-key nibbles (Figure 24). The number of involved S-boxes in the outer rounds is $7 + 12 = 19$ (Figure 22), so the time complexity of the online phase is $2^{4\times19} \times 2^{4\times1} \times \frac{7+12}{20\times8} \approx 2^{76.92}$, slightly lower than in [62]. The 76-bit subkey space is reduced by 4 bits. The data complexity is $2^{4\times8} = 2^{32}$, since 8 input nibbles are active (Figure 22).

9.2 $\mathcal{DS}$-MITM Attack on SKINNY-128-256

To show the usefulness of our method for ciphers with linear key schedules, we apply it to find a $\mathcal{DS}$-MITM attack on 19 rounds of SKINNY-128-256 in the single-tweakey setting. Although there is a 9.5-round $\mathcal{DS}$-MITM distinguisher on SKINNY-128-256 which we can extend to a key recovery attack on 19 rounds, we discovered that using an 8.5-round distinguisher yields a better attack in terms of time complexity and memory complexity.

Complexity For distinguisher part (Figure 25), 30 words should be guessed to determine the output sequence. Hence, in the offline phase the time complexity is $2^{8\times 25} \times 2^{8\times 1} \times \frac{25}{16\times 19} C_E \approx 2^{204.39} C_E$, and the memory complexity is $(2^8 - 1) \times 8 \times 1 \times 2^{8\times 25} \approx 2^{210.99}$ bits. 12 cells are active in the input state (Figure 26), so the data complexity is $2^{8\times 12} = 2^{96}$. For the online phase (Figure 27), 45 sub-key bytes are involved in the key recovery, but thanks to the key-bridging technique they can be deduced from 29 sub-key bytes. In total, $22 + 69 = 91$ S-boxes are involved in the outer rounds (Figure 26), so the time complexity of the online phase is $2^{29\times 8} \times 2^{8\times 1} \times \frac{22+69}{16\times 19} \times C_E \approx 2^{238.26} C_E$, where C_E is the running time of 19 rounds of SKINNY-128-256.

While we demonstrated the application of our method only for $\mathcal{DS}$-MITM attacks, it can be straightforwardly applied to find key-recovery-friendly distinguishers in linear or differential cryptanalysis as well.

9.3 $\mathcal{DS}$-MITM Attack on SKINNY-64-192

Applying our searching method to find a $\mathcal{DS}$-MITM attack for SKINNY-64-192, we discovered a 21-round $\mathcal{DS}$-MITM attack in the single-tweakey setting relying on 8.5-round distinguisher. Again, although there is a 9.5-round $\mathcal{DS}$-MITM distinguisher for SKINNY-64-192 which can be used to construct a 21-round attack, thanks to our new model we noticed that building the attack on an 8.5-round distinguisher results in a better attack in terms of complexity.

Complexity Figure 29 illustrates the distinguisher of our attack on 21 rounds of SKINNY-64-192, where 31 nibbles should be guessed in the offline phase. Hence, the time complexity of the offline phase is $2^{4\times 31} \times 2^{4\times 2} \times \frac{31}{21\times 16} C_E \approx 2^{128.56} C_E$, and the memory complexity is $(2^{4\times 2} - 1) \times 4 \times 2^{4\times 31} \approx 2^{133.99}$ bits. As it is shown in Figure 30, 15 nibbles are active in the input state in the first round, which shows that the data complexity of our attack is $2^{4\times 15} = 2^{60}$ chosen plaintexts. Figure 31 represents that 63 sub-key nibbles are involved in the key recovery attack, but as it can be seen in Figure 32 they can be deduced from only 45 sub-key nibbles. As a result, the time complexity of the online phase is $2^{45\times 4} \times 2^{4\times 2} \times \frac{29+101}{21\times 16} C_E \approx 2^{186.63} C_E$.

9.4 $\mathcal{DS}$-MITM Attack on SKINNY-64-128

We also applied our method to find an 18-round $\mathcal{DS}$-MITM attack on SKINNY-64-128, which relies on a 7.5-round distinguisher rather than an 8.5-round distinguisher.

Complexity We use a 7.5-round distinguisher as illustrated in Figure 33 in our attack on 18 rounds of SKINNY-64-128. According to Figure 33, 14 nibbles should be guessed in the offline phase. Hence, the time complexity of the offline phase is $2^{4\times14} \times 2^{4\times1} \times \frac{14}{18\times16} C_E \approx 2^{55.64} C_E$, and the memory complexity of this phase is $(2^{4\times1} - 1) \times 4 \times 1 \times 2^{4\times14} \approx 2^{61.91}$bits. As it can be seen in Figure 35, 47 sub-key nibbles are involved in our attack. However, as it can be seen in Figure 36 only 31 sub-key nibbles should be actually guessed. As a result the time complexity of the online phase is $2^{4\times31} \times 2^{4\times1} \times \frac{16+74}{18\times16} C_E \approx 2^{126.32} C_E$. By the way, since 8 nibbles are active in the input of round 0 in Figure 34, the data complexity is $2^{4\times8} = 2^{32}$ chosen plaintexts.

10 Comparison Between Solvers and Limitations

Throughout our experiments, we observed that in many cases, the SAT-based method outperforms the other methods when we want to find only a feasible solution. For instance, thanks to the SAT-based method implemented in Autoguess, and executing on a single core Intel Core i9 processor at 3.6 GHz, it takes less than a second to reproduce the GD attack on `Enocoro-128v2` in [14], whereas finding the same result (i.e., a feasible solution) via the MILP-based method takes at least couple of minutes. We had the same observation in applying Autoguess to reproduce the best previous GD attacks on `SNOW1`, `SNOW2`, `SNOW3`, `KCipher2`, etc. However, we observed that in some other applications, using the Gröbner basis approach gives a better performance. For instance, we observed that the Gröbner basis-based method performs very well in our applications for the key-bridging technique, while it also ensures the optimum output. Therefore, it makes sense to integrate different encoding methods in one tool, since each encoding method might be suitable for a specific application. Although our tool is easy to use, it is very general, and hence it may produce attacks slower than the dedicated attacks designed for a specific cipher. We would like to discuss some of its main limitations and directions for future work as follows:

- Autoguess currently does not support automatically finding solutions in which a function (or group) of variables is guessed rather than guessing every single variable. For example, it is not possible to find a solution in which $x \oplus y$ is guessed unless we define a dummy variable such as $d = x \oplus y$ in the system of connection relations.
- The choice of a new variable to guess depends only on previously guessed variables and ignores the guessed values, whereas considering the values during the guessing process might result in a guess-and-determine attack with less complexity (static guessing strategy vs. dynamic guessing strategy).
- Guessing some variables might result in a system of linear equations in the middle steps of knowledge propagation which causes some other variables to be determined for free. However, we are currently unable to check the existence of such systems of linear equations in our method.
- In some cases, the derived SAT or MILP models are too heavy to solve in a reasonable time. For instance, applying Autoguess to bit-oriented ciphers

with large block sizes such as `Ascon` [24] yields a very heavy SAT or MILP models.

11 Conclusion

In this paper, we proposed Autoguess, an easy-to-use, general and open-source tool to search for a minimal guess basis in guess-and-determine attacks which is also applicable to find key-bridges in key recovery attacks on block ciphers. As a new encoding method, we introduced the SAT/SMT encoding of the guess-and-determine problem which outperforms the MILP and Gröbner basis approaches from the performance point of view particularly when searching for feasible solutions. Our tool integrates the new SAT/SMT encoding method, as well as MILP- and Gröbner basis-based methods to automate guess-and-determine and key-bridging techniques. To demonstrate the usefulness of our tool, we showed that many previous guess-and-determine attacks as well as key-bridges, which were usually discovered based on tedious and error-prone manual approaches, can now simply be discovered in a couple of seconds using our tool. Moreover, using our tool, we could improve several of the previous results regarding key-bridging technique and guess-and-determine attack. This demonstrates the potential of using Autoguess for both design and cryptanalysis. Moreover, we managed to integrate our CP-based approach for key-bridging technique into the previous CP-based frameworks for finding distinguishers and introduced a general CP model to search for key recovery friendly distinguishers supporting both linear and nonlinear key schedules for the first time.

Acknowledgments

The authors would like to thank Mohammad Ali Orumiehchiha and Siwei Sun for motivating discussions and some helpful comments on the earlier version of the tool.

References

1. Ahmadi, H., Eghlidos, T.: Heuristic guess-and-determine attacks on stream ciphers. IET Information Security **3**(2), 66–73 (2009)
2. Ankele, R., Dobraunig, C., Guo, J., Lambooij, E., Leander, G., Todo, Y.: Zero-correlation attacks on tweakable block ciphers with linear tweakey expansion. IACR Transactions on Symmetric Cryptology **2019**(1), 192–235 (Mar 2019). `https://doi.org/10.13154/tosc.v2019.i1.192-235`, `https://tosc.iacr.org/index.php/ToSC/article/view/7402`
3. Asín, R., Nieuwenhuis, R., Oliveras, A., Rodríguez-Carbonell, E.: Cardinality networks and their applications. In: Theory and Applications of Satisfiability Testing – SAT 2009. pp. 167–180. Springer (2009)

4. Babbage, S., De Cannière, C., Lano, J., Preneel, B., Vandewalle, J.: Cryptanalysis of SOBER-t32. In: Johansson, T. (ed.) Fast Software Encryption – FSE 2003. LNCS, vol. 2887, pp. 111–128. Springer (2003). https://doi.org/10.1007/978-3-540-39887-5_10
5. Barrett, C., Conway, C.L., Deters, M., Hadarean, L., Jovanović, D., King, T., Reynolds, A., Tinelli, C.: CVC4. In: Computer Aided Verification – CAV 2011. pp. 171–177. Springer (2011), available from https://github.com/CVC4/CVC4
6. Beierle, C., Jean, J., Kölbl, S., Leander, G., Moradi, A., Peyrin, T., Sasaki, Y., Sasdrich, P., Sim, S.M.: The SKINNY family of block ciphers and its low-latency variant MANTIS. In: Advances in Cryptology – CRYPTO 2016. pp. 123–153. Springer (2016)
7. Beierle, C., Leander, G., Moradi, A., Rasoolzadeh, S.: CRAFT: lightweight tweakable block cipher with efficient protection against DFA attacks. IACR Trans. Symmetric Cryptol. **2019**(1), 5–45 (2019). https://doi.org/10.13154/tosc.v2019.i1.5-45
8. Biere, A.: Lingeling, Plingeling and Treengeling entering the SAT competition 2013. Proceedings of SAT competition **2013**, 1 (2013), available from https://github.com/arminbiere/lingeling
9. Biere, A., Fazekas, K., Fleury, M., Heisinger, M.: CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In: Balyo, T., Froleyks, N., Heule, M., Iser, M., Järvisalo, M., Suda, M. (eds.) SAT Competition 2020 – Solver and Benchmark Descriptions. Department of Computer Science Report Series B, vol. B-2020-1, pp. 51–53. University of Helsinki (2020)
10. Biryukov, A., Derbez, P., Perrin, L.: Differential analysis and meet-in-the-middle attack against round-reduced TWINE. In: Fast Software Encryption – FSE 2015. pp. 3–27. Springer (2015)
11. Bouillaguet, C., Derbez, P., Fouque, P.A.: Automatic search of attacks on round-reduced AES and applications. In: Advances in Cryptology – CRYPTO 2011. pp. 169–187. Springer (2011)
12. Brickenstein, M., Dreyer, A.: PolyBoRi: A framework for Gröbner-basis computations with Boolean polynomials. Journal of Symbolic Computation **44**(9), 1326 – 1345 (2009). https://doi.org/DOI:10.1016/j.jsc.2008.02.017, effective Methods in Algebraic Geometry. Available from http://polybori.sourceforge.net
13. Bruttomesso, R., Cimatti, A., Franzén, A., Griggio, A., Sebastiani, R.: The MathSAT 4 SMT solver. In: Computer Aided Verification – CAV 2008. pp. 299–303. Springer (2008), available from https://mathsat.fbk.eu/
14. Cen, Z., Feng, X., Wang, Z., Cao, C.: Minimizing deduction system and its application. arXiv preprint arXiv:2006.05833 (2020), https://arxiv.org/abs/2006.05833
15. Chen, Q., Shi, D., Sun, S., Hu, L.: Automatic Demirci-Selçuk meet-in-the-middle attack on SKINNY with key-bridging. In: International Conference on Information and Communications Security – ICICS 2019. pp. 233–247. Springer (2019)
16. Cook, S.A.: The complexity of theorem-proving procedures. In: Symposium on Theory of computing – STOC 1971. pp. 151–158 (1971)
17. Courtois, N., Klimov, A., Patarin, J., Shamir, A.: Efficient algorithms for solving overdefined systems of multivariate polynomial equations. In: Advances in Cryptology – EUROCRYPT 2000. pp. 392–407. Springer (2000)
18. Cplex, I.I.: V12.10.0: User's manual for cplex (2020), available from https://www.ibm.com/analytics/cplex-optimizer
19. Cui, Y., Xu, H., Qi, W.: Improved integral attacks on 24-round LBlock and LBlock-s. IET Information Security (2020)

20. Daemen, J., Rijmen, V.: AES proposal: Rijndael (1999), https://csrc.nist.gov/csrc/media/projects/cryptographic-standards-and-guidelines/documents/aes-development/rijndael-ammended.pdf
21. Danner, J., Kreuzer, M.: A fault attack on KCipher-2. International Journal of Computer Mathematics: Computer Systems Theory pp. 1–22 (2020)
22. De Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: Tools and Algorithms for the Construction and Analysis of Systems – TACAS 2008. pp. 337–340. Springer (2008), available from https://github.com/Z3Prover/z3
23. Decker, W., Greuel, G.M., Pfister, G., Schönemann, H.: Singular 4-2-0 — A computer algebra system for polynomial computations. http://www.singular.uni-kl.de (2020)
24. Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: Ascon v1.2. Submission to the CAESAR Competition (2016)
25. Dunkelman, O., Keller, N.: The effects of the omission of last round's MixColumns on AES. Information Processing Letters **110**(8-9), 304–308 (2010)
26. Dunkelman, O., Keller, N., Shamir, A.: Improved single-key attacks on 8-round AES-192 and AES-256. In: Advances in Cryptology – ASIACRYPT 2010. pp. 158–176. Springer (2010)
27. Dutertre, B.: Yices 2.2. In: Computer Aided Verification – CAV 2014. pp. 737–744. Springer (2014), available from https://yices.csl.sri.com/
28. Een, N.: MiniSat: A SAT solver with conflict-clause minimization. In: Theory and Applications of Satisfiability Testing – SAT 2005. pp. 502–518 (2005), available from http://minisat.se
29. Ellson, J., Gansner, E., Koutsofios, L., North, S.C., Woodhull, G.: Graphviz – open source graph drawing tools. In: Graph Drawing – GD 2001. pp. 483–484. Springer (2001), https://graphviz.org
30. ETSI/SAGE: Specification of the 3gpp confidentiality and integrity algorithms 128-EEA3 and 128-EIA3: ZUC specification. ETSI/SAGE, Document 2, Version 1.6 (2011)
31. Faugere, J.C.: A new efficient algorithm for computing Gröbner bases without reduction to zero (F5). In: Proceedings of the 2002 international symposium on Symbolic and algebraic computation. pp. 75–83 (2002)
32. Ganesh, V., Liang, J.: MapleSAT (2017), available from https://sites.google.com/a/gsd.uwaterloo.ca/maplesat/maplesat
33. Gario, M., Micheli, A.: PySMT: a solver-agnostic library for fast prototyping of SMT-based algorithms. In: SMT Workshop 2015 (2015), available from https://github.com/pysmt/pysmt
34. Gecode Team: Gecode: Generic constraint development environment (2006), available from http://www.gecode.org
35. Golić, J.D.: Cryptanalysis of alleged A5 stream cipher. In: Fumy, W. (ed.) Advances in Cryptology – EUROCRYPT '97. LNCS, vol. 1233, pp. 239–255. Springer (1997). https://doi.org/10.1007/3-540-69053-0_17
36. Guan, J., Ding, L., Liu, S.: Guess and determine attack on SNOW3G and ZUC. Journal of Software **24**(6), 1324–1333 (2013)
37. Gurobi Optimization, Incorporate: Gurobi optimizer reference manual (2020), available from https://www.gurobi.com
38. Gutiérrez, A.F., Naya-Plasencia, M.: Improving key-recovery in linear attacks: Application to 28-round PRESENT. In: Advances in Cryptology – EUROCRYPT 2020. LNCS, vol. 12105. Springer (2020). https://doi.org/10.1007/978-3-030-45721-1_9

39. Hadarean, L., Hyvarinen, A., Niemetz, A., Reger, G.: 14th international satisfiability modulo theories competition (SMT-COMP 2019): Rules and procedures
40. Hawkes, P., Rose, G.G.: Guess-and-determine attacks on SNOW. In: Selected Areas in Cryptography – SAC 2002. pp. 37–46. Springer (2002)
41. Horáček, J., Kreuzer, M.: On conversions from CNF to ANF. Journal of Symbolic Computation **100**, 164–186 (2020)
42. Ignatiev, A., Morgado, A., Marques-Silva, J.: PySAT: A Python toolkit for prototyping with SAT oracles. In: Theory and Applications of Satisfiability Testing – SAT 2018. pp. 428–437 (2018), available from `https://pysathq.github.io/docs/html/api/solvers.html`
43. Jean, J.: TikZ for Cryptographers. `https://www.iacr.org/authors/tikz/` (2016)
44. Joux, A., Vitse, V.: A crossbred algorithm for solving Boolean polynomial systems. In: International Conference on Number-Theoretic Methods in Cryptology. pp. 3–21. Springer (2017)
45. Khazaei, S., Moazami, F.: On the computational complexity of finding a minimal basis for the guess and determine attack. ISeCure – The ISC International Journal of Information Security **9**(2), 101–110 (2017)
46. Knudsen, L.R., Meier, W., Preneel, B., Rijmen, V., Verdoolaege, S.: Analysis methods for (alleged) RC4. In: Ohta, K., Pei, D. (eds.) Advances in Cryptology – ASIACRYPT '98. LNCS, vol. 1514, pp. 327–341. Springer (1998). `https://doi.org/10.1007/3-540-49649-1_26`
47. Kolay, S., Mukhopadhyay, D.: Khudra: a new lightweight block cipher for FPGAs. In: Security, Privacy, and Applied Cryptography Engineering – SPACE 2014. pp. 126–145. Springer (2014)
48. Lazard, D.: Gröbner bases, Gaussian elimination and resolution of systems of algebraic equations. In: European Conference on Computer Algebra. pp. 146–156. Springer (1983)
49. Lin, L., Wu, W., Zheng, Y.: Automatic search for key-bridging technique: applications to LBlock and TWINE. In: Fast Software Encryption – FSE 2016. pp. 247–267. Springer (2016)
50. Liu, Y., Wang, Q., Rijmen, V.: Automatic search of linear trails in ARX with applications to SPECK and Chaskey. In: Applied Cryptography and Network Security – ACNS 2016. pp. 485–499. Springer (2016)
51. Marques-Silva, J.P., Sakallah, K.A.: GRASP: A search algorithm for propositional satisfiability. IEEE Transactions on Computers **48**(5), 506–521 (1999)
52. Martins, R., Joshi, S., Manquinho, V., Lynce, I.: Incremental cardinality constraints for MaxSAT. In: Principles and Practice of Constraint Programming – CP 2014. pp. 531–548. Springer (2014)
53. Mitchell, S., OSullivan, M., Dunning, I.: PuLP: a linear programming toolkit for Python. The University of Auckland, Auckland, New Zealand (2011), available from `https://github.com/coin-or/pulp`
54. Nethercote, N., Stuckey, P.J., Becket, R., Brand, S., Duck, G.J., Tack, G.: MiniZinc: Towards a standard CP modelling language. In: Principles and Practice of Constraint Programming – CP 2007. pp. 529–543. Springer (2007)
55. Niemetz, A., Preiner, M., Biere, A.: Boolector 2.0 system description. Journal on Satisfiability, Boolean Modeling and Computation **9**, 53–58 (2014 (published 2015))
56. Niemetz, A., Preiner, M., Biere, A.: Boolector 2.0 system description. Journal on Satisfiability, Boolean Modeling and Computation **9**, 53–58 (2014 (published 2015))
57. Ogawa, T., Liu, Y., Hasegawa, R., Koshimura, M., Fujita, H.: Modulo based CNF encoding of cardinality constraints and its application to MaxSAT solvers. In:

International Conference on Tools with Artificial Intelligence – ICTAI 2013. pp. 9–17. IEEE (2013)
58. Özen, M., Çoban, M., Karakoç, F.: A guess-and-determine attack on reduced-round Khudra and weak keys of full cipher. IACR Cryptol. ePrint Arch. **2015**, 1163 (2015)
59. Perron, L., Furnon, V.: OR-Tools (2021), `https://developers.google.com/optimization/`
60. Prud'homme, C., Fages, J.G., Lorca, X.: Choco Solver Documentation. TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S. (2016), `http://www.choco-solver.org`
61. Qin, L., Dong, X., Wang, X., Jia, K., Liu, Y.: Automated Search Oriented to Key Recovery on Ciphers with Linear Key Schedule: Applications to Boomerangs in SKINNY and ForkSkinny. IACR Transactions on Symmetric Cryptology **2021**(2), 249–291 (Jun 2021). `https://doi.org/10.46586/tosc.v2021.i2.249-291`, `https://tosc.iacr.org/index.php/ToSC/article/view/8911`
62. Shi, D., Sun, S., Derbez, P., Todo, Y., Sun, B., Hu, L.: Programming the Demirci-Selçuk meet-in-the-middle attack with constraints. In: Advances in Cryptology – ASIACRYPT 2018. pp. 3–34. Springer (2018)
63. Siegenthaler, T.: Decrypting a class of stream ciphers using ciphertext only. IEEE Transactions on Computers **34**(1), 81–85 (1985). `https://doi.org/10.1109/TC.1985.1676518`
64. Sinz, C.: Towards an optimal CNF encoding of Boolean cardinality constraints. In: Principles and Practice of Constraint Programming – CP 2005. pp. 827–831. Springer (2005)
65. Sun, L., Wang, W., Wang, M.: More accurate differential properties of LED64 and Midori64. IACR Transactions on Symmetric Cryptology **2018**(3), 93–123 (Sep 2018). `https://doi.org/10.13154/tosc.v2018.i3.93-123`, `https://tosc.iacr.org/index.php/ToSC/article/view/7298`
66. Team, Z.D.: The ZUC-256 stream cipher (2018), `http://www.is.cas.cn/ztzl2016/zouchongzhi/201801/W020180126529970733243.pdf`
67. The Sage Developers: SageMath, the Sage Mathematics Software System (Version 9.2.0) (2021), `https://www.sagemath.org`
68. Wang, Y., Wu, W., Yu, X., Zhang, L.: Security on LBlock against biclique cryptanalysis. In: Workshop on Information Security Applications – WISA 2012. pp. 1–14. Springer (2012)
69. Wu, W., Zhang, L.: LBlock: A lightweight block cipher. In: Lopez, J., Tsudik, G. (eds.) Applied Cryptography and Network Security – ACNS 2011. pp. 327–344. Springer (2011)
70. Zaikin, O., Kochemazov, S.: An improved SAT-based guess-and-determine attack on the alternating step generator. In: Information Security Conference – ISC 2017. pp. 21–38. Springer (2017)

— Auxiliary Material —

A Brute-force Search to Find a Guess Basis

Algorithm 4: `Propagate`

```
Input: Set $K$ of variables, and set $\mathcal{R}$ of connection relations over $X \supseteq K$
Output: Set $Y = \texttt{Propagate}(K)$
1  Initialize a dictionary KnownVars of length $n$, where, $keys(\texttt{KnownVars}) = X$;
2  for $v \in X$ do
3      if $v \in K$ then
4          KnownVars[$v$] $\leftarrow 1$;
5      else
6          KnownVars[$v$] $\leftarrow 0$;
7  $Y \leftarrow \emptyset$;
8  while $Y \neq K$ do
9      $Y \leftarrow K$;
10     for $r \in \mathcal{R}$ do
11         if $r$ is a symmetric relation then
12             if $\sum_{v \in r} \textit{KnownVars}[v] = |r| - 1$ then
13                 DeterminedVar $\leftarrow v \in r$ for which KnownVars[$v$] $= 0$;
14                 $K \leftarrow K \cup \{\texttt{DeterminedVar}\}$;
15                 KnownVars[DeterminedVar] $\leftarrow 1$;
16         if $r$ is an implication relation then
17             if $\bigwedge_{v \in \textit{LHS}(r)} v = 1$ then
18                 DeterminedVar $\leftarrow v \in \texttt{RHS}(r)$;
19                 $K \leftarrow K \cup \texttt{RHS}(r)$;
20                 KnownVars[DeterminedVar] $\leftarrow 1$;
21 return Y;
```

Algorithm 5: `Exhaustive Search for a Minimal Guess Basis`

```
Input: Set $X$ of variables and set $\mathcal{R}$ of connection relations over $X$
Output: A minimal guess basis $\mathbb{G}$
1  $n \leftarrow |X|$;
2  for $k = 1 \rightarrow n$ do
3      $P \leftarrow \{S : S \subseteq X, |S| = k\}$;
4      for $S \in P$ do
5          if $\textit{Propagate}(S) = X$ then
6              return $\mathbb{G} = S$
```

B Automatic GD Attack on CRAFT

CRAFT [7] is a lightweight tweakable block cipher which receives a 64-bit plaintext with a 128-bit master key plus a 64-bit master tweak and then perform an SPN round function as shown in Figure 14 for 32 times to produce a 64-bit ciphertext. To produce the sub-tweakeys, the tweakey schedule of CRAFT splits the 128-bit key K of CRAFT into two halves K_0 and K_1 at first and then using the nibble-wise permutation Q, it mixes the key K with the given master tweak T according to the following relations to produce four different tweakeys:

$$TK_0 = K_0 + T,\ TK_1 = K_1 + T,\ TK_2 = K_0 + Q(T),\ TK_3 = K_0 + Q(T),$$

where $Q = [12, 10, 15, 5, 14, 8, 9, 2, 11, 3, 7, 4, 6, 0, 1, 13]$. After that, starting from TK_0, CRAFT uses the four derives tweakeys TK_0, TK_1, TK_2, TK_3 periodically as the sub-tweakey in each round.

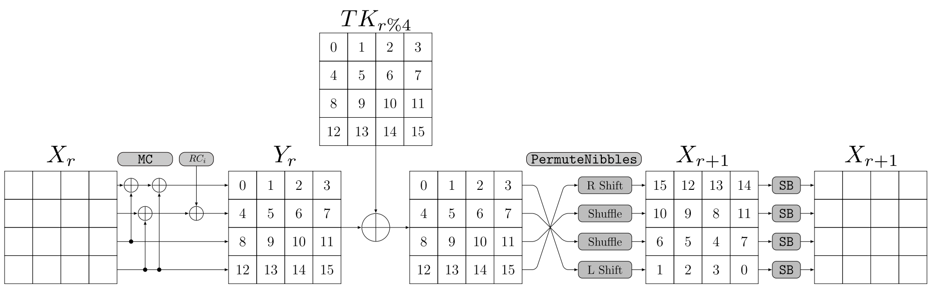


Fig. 14: The Round Function of CRAFT

Here, given one (or at most two) known plaintext/ciphertext pair(s), we look for a word-oriented GD attack on reduced-round CRAFT. As it is represented in Figure 14, let X_r and Y_r denote the input and output of MixColumn layer of round r respectively. Besides, to represent the ith nibble of X_r and Y_r we use $X_{r,i}$ and $Y_{r,i}$ respectively. According to Figure 14 and taking into account that the master tweak T is publicly known, the connection relations corresponding to R rounds of CRAFT are as follows:

$$\begin{aligned}
&[X_{r,i}, Y_{r,i+8}, X_{r,i+12}, Y_{r,i}] && \text{for } 0 \le i \le 3,\\
&[X_{r,i+4}, Y_{r,i+12}, Y_{r,i+4}] && \text{for } 0 \le i \le 3,\\
&[X_{r,i+8}, Y_{r,i+8}] && \text{for } 0 \le i \le 3,\\
&[X_{r,i+12}, Y_{r,i+12}] && \text{for } 0 \le i \le 3,\\
&[Y_{r,i}, K_{r\%2,i}, X_{r+1,P[i]}] && \text{for } 0 \le i \le 15,
\end{aligned}$$

where $r\%2$ denote r modulo 2, $0 \le r \le R$, and P is the nibble-wise permutation used in each round of CRAFT. Note that X_0, and X_R both are known as they are corresponding to the plaintext and ciphertext, respectively. Table 4 briefly describes the result of our nibble-wise GD attack on CRAFT. We also couldn't find a nibble-wise GD attack on 14 rounds of CRAFT. In conclusion, at least 14 rounds of CRAFT are required to mix all the key nibbles into the cipher.

Table 4: Number of guessed variables in GD attack on 1 to 13 rounds of CRAFT

#Rounds	1	2	3	4	5	6	7	8	9	10	11	12	13
#Guessed nibbles	0	16	16	20	20	23	23	26	26	28	28	29	30
#Required data (KP)	1	2	2	2	2	2	2	2	2	2	2	2	2

C Automatic GD Attack on SKINNY

We also applied our tool on SKINNY to search for word-oriented GD attacks on reduced-round of this cipher. As it is illustrated in Figure 15, let X_r and Y_r denote the internal state before SB and after ART layers respectively. Besides, according to the tweakey schedule of SKINNY-n-n, the ith cell of round tweakey TK_r is equal to $P_T^{(r)}[i]$th cell of $TK1$. Hence, the connection relations corresponding to R rounds of SKINNY-n-n are as follows:

$$\begin{aligned}
&[X_r[i], TK1[P_T^{(r)}[i]], Y_r[i]] && \text{for } 0 \le i \le 7,\\
&[X_r[i], Y_r[i]] && \text{for } 8 \le i \le 15,\\
&[Y_r[P[i]], Y_r[P[i+8]], Y_r[P[i+12]], X_{r+1}[i]] && \text{for } 0 \le i \le 3,\\
&[Y_r[P[i]], X_{r+1}[P[i+4]]] && \text{for } 0 \le i \le 3,\\
&[Y_r[P[i+4]], Y_r[P[i+8]], X_{r+1}[i+8]] && \text{for } 0 \le i \le 3,\\
&[Y_r[P[i]], Y_r[P[i+8]], X_{r+1}[i+12]] && \text{for } 0 \le i \le 3,
\end{aligned}$$

where P is the permutation of SKINNY's round function which is performed on the position of cells and defined as follows:

$$P = [0, 1, 2, 3, 7, 4, 5, 6, 10, 11, 8, 9, 13, 14, 15, 12].$$

Given one (or at most two) known plaintext/ciphertext pair(s) we are interested to find the minimum number of guessed words such that all of the involved sub-tweakeys can be deduced. In a similar way, we can discover GD attacks on the other variants of SKINNY. Table 5, summarizes our results for GD attack on SKINNY. As it can be seen our attacks reach up to 11 rounds of this cipher.

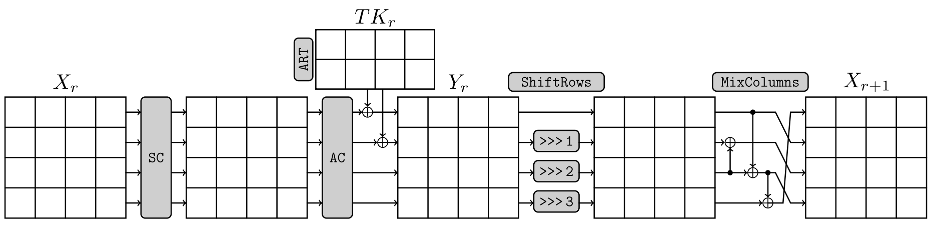


Fig. 15: Variables used in GD attack on SKINNY-n-n

Table 5: Number of guessed variables in GD attack on 1 to 11 rounds of SKINNY

Cipher	#Rounds	1	2	3	4	5	6	7	8	9	10	11
SKINNY-n-n	#Guessed variables	0	0	3	5	6	9	10	12	12	14	15
	#Required data (KP)	1	1	1	1	1	1	1	1	1	1	1
SKINNY-n-$2n$	#Guessed variables	8	16	19	21	22	25	26	28	28	30	31
	#Required data (KP)	1	2	2	2	2	2	2	2	2	2	2
SKINNY-n-$3n$	#Guessed variables	16	32	35	37	38	41	42	44	44	46	47
	#Required data (KP)	2	2	2	2	2	2	2	2	2	2	2

D Key Schedule of PRESENT-128

Algorithm 6: Key schedule of PRESENT-128

```
Input: A master key $K = \kappa_0 \cdots \kappa_{127}$ of 128 bits, a number of rounds $r$
       where $r \leq 31$
Output: $r+1$ round sub-keys $K_i$ of 64 bits
1 $K_0 \leftarrow \kappa_0 \cdots \kappa_{63}$;    ▷ Extract first round sub-key;
2 for $i = 1 \rightarrow r$ do
3     $\kappa_0 \ldots \kappa_{127} \leftarrow \kappa_{61} \ldots \kappa_{127}\kappa_0 \ldots \kappa_{60}$; ▷ Rotate 61 bits to the left;
4     $(\kappa_0\kappa_1\kappa_2\kappa_3) \leftarrow S(\kappa_0\kappa_1\kappa_2\kappa_3)$;
5     $(\kappa_4\kappa_5\kappa_6\kappa_7) \leftarrow S(\kappa_4\kappa_5\kappa_6\kappa_7)$;    ▷ Apply S-box on two leftmost
       nibbles;
6     $\kappa_{61}\kappa_{62}\kappa_{63}\kappa_{64}\kappa_{65} \leftarrow \kappa_{61}\kappa_{62}\kappa_{63}\kappa_{64}\kappa_{65} \oplus i$;    ▷ Add round counter;
7     $K_i \leftarrow \kappa_0 \ldots \kappa_{63}$;    ▷ Extract round sub-key;
8 return $\{K_i\}_{i=0}^{r}$;
```

E Key Schedule of LBlock

Algorithm 7: Key schedule of `LBlock`

```
Input: A master key $K = \kappa_0 \cdots \kappa_{79}$ of 79 bits, a number of rounds $r$
       where $0 \leq r \leq 31$
Output: $r + 1$ sub-keys $K_i$ of 32 bits
1 $K_0 \leftarrow \kappa_0 \cdots \kappa_{31}$;          ▷ Extract first round sub-key;
2 for $i = 1 \rightarrow r$ do
3     $\kappa_0 \dots \kappa_{79} \leftarrow \kappa_{29} \dots \kappa_{79}\kappa_0 \dots \kappa_{28}$;  ▷ Rotate 29 bits to the left;
4     $(\kappa_0\kappa_1\kappa_2\kappa_3) \leftarrow S_9(\kappa_0\kappa_1\kappa_2\kappa_3)$;
5     $(\kappa_4\kappa_5\kappa_6\kappa_7) \leftarrow S_8(\kappa_4\kappa_5\kappa_6\kappa_7)$;     ▷ Apply S-box on two leftmost
      nibbles;
6     $\kappa_{29}\kappa_{30}\kappa_{31}\kappa_{32}\kappa_{33} \leftarrow \kappa_{29}\kappa_{30}\kappa_{31}\kappa_{32}\kappa_{33} \oplus i$;   ▷ Add round counter;
7     $K_i \leftarrow \kappa_0 \dots \kappa_{31}$;              ▷ Extract round sub-key;
8 return $\{K_i\}_{i=0}^{r}$;
```

F Determination of Key Bits Involved in Key Recovery of Integral Attack on 24 Rounds of LBlock

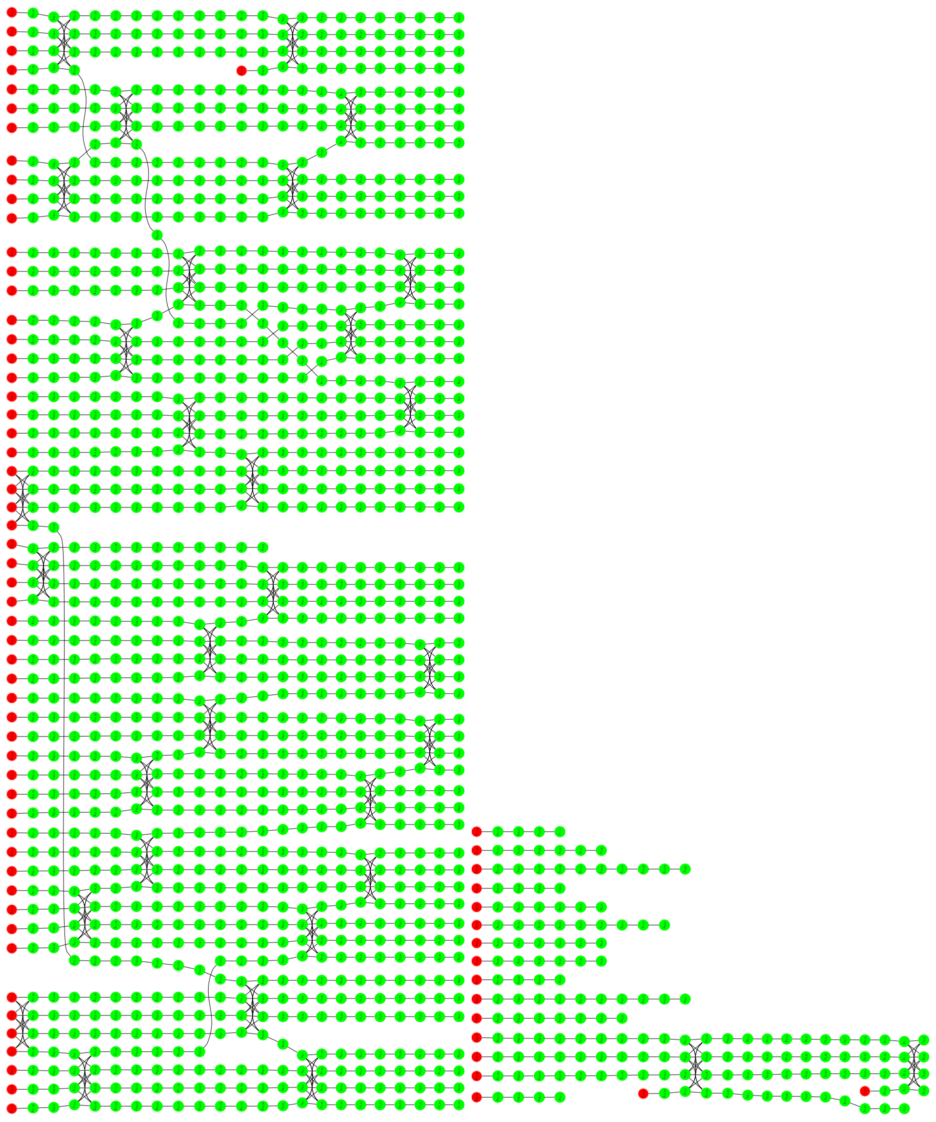


Fig. 16: Determination flow in key recovery of impossible differential attack on 23 rounds of LBlock

G Determination of GD Attack on AES

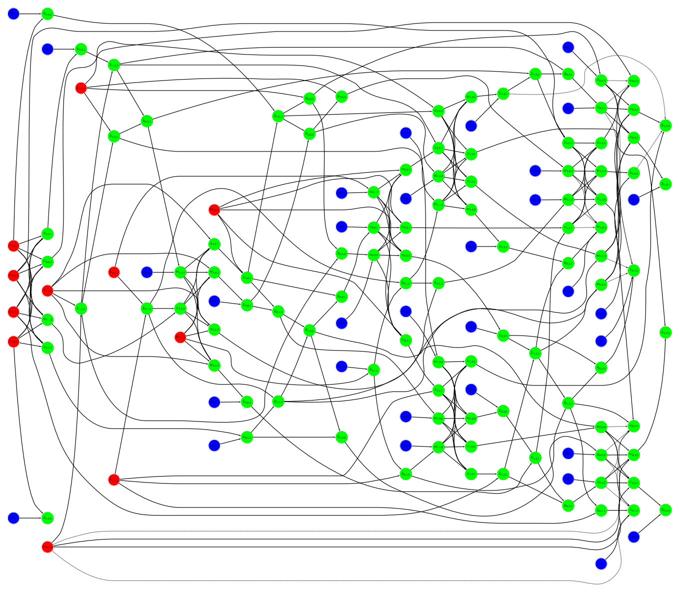


Fig. 17: Determination flow in GD attack with time complexity of 2^{88} on two rounds of AES which required only one known plaintext/ciphertext pair

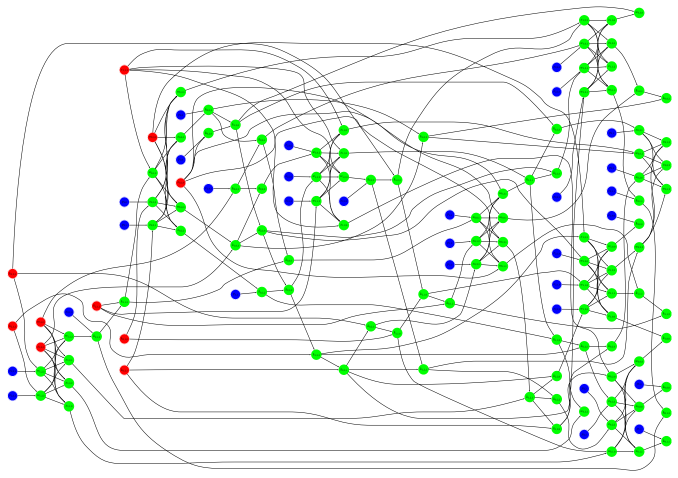


Fig. 18: Determination flow in GD attack with time complexity of 2^{80} on two rounds of AES which required only one known plaintext/ciphertext pair

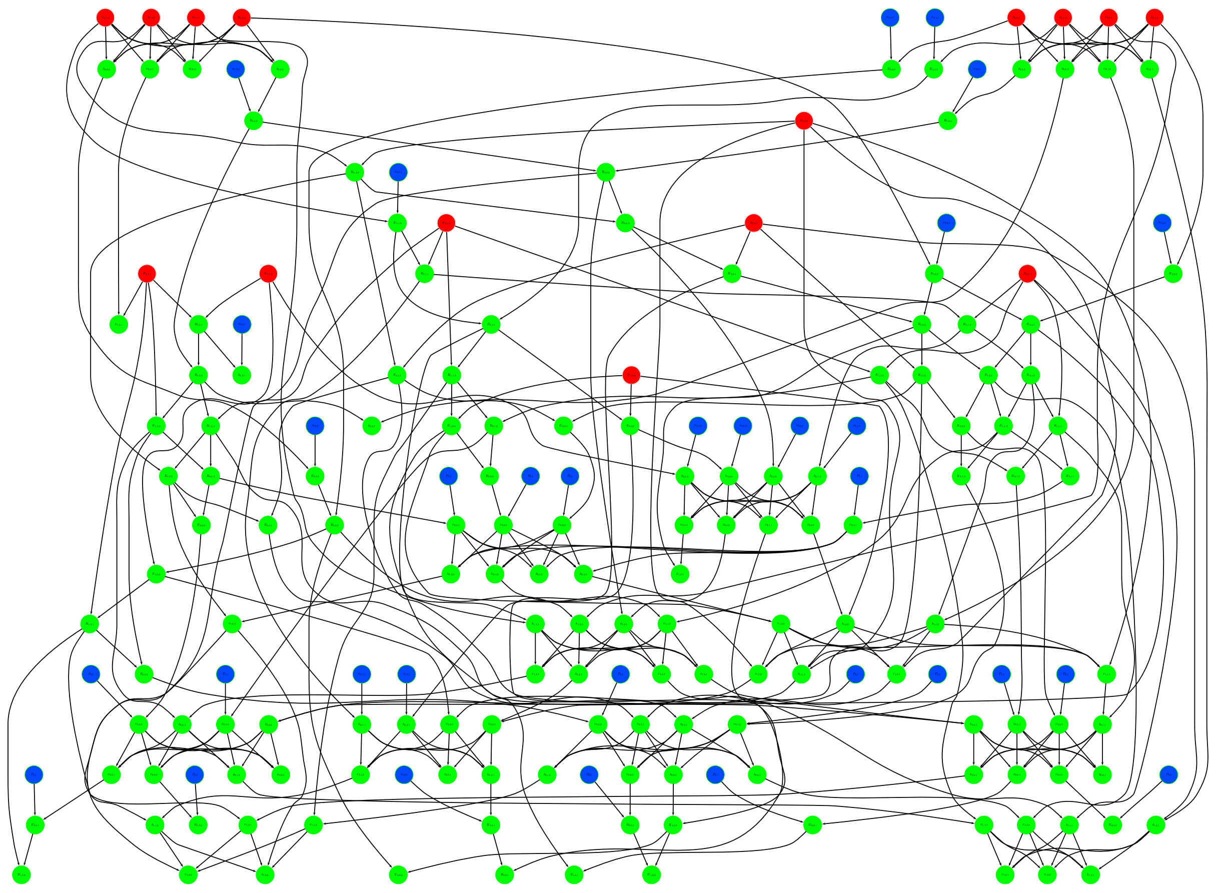
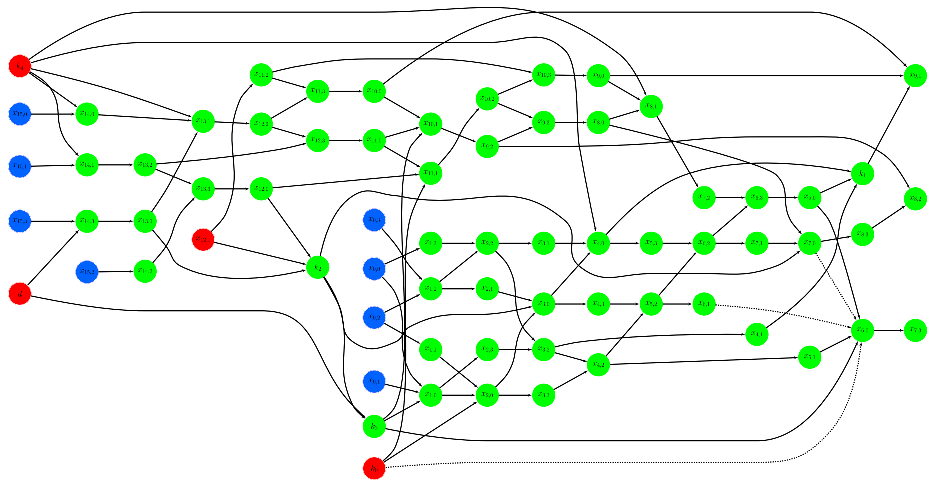


Fig. 19: Determination flow in GD attack on three rounds of AES

H GD Attack on Khudra



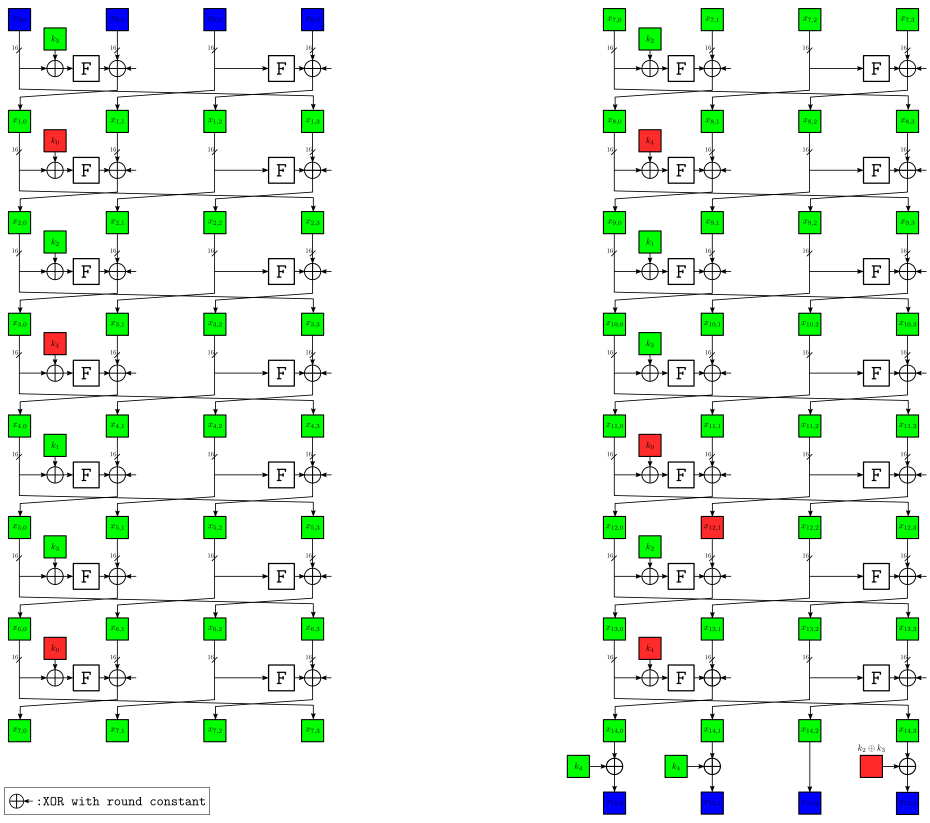


Fig. 20: Guess-and-determine attack on 14 rounds of Khudra

I $\mathcal{DS}$-MITM attack on 20-round TWINE-80

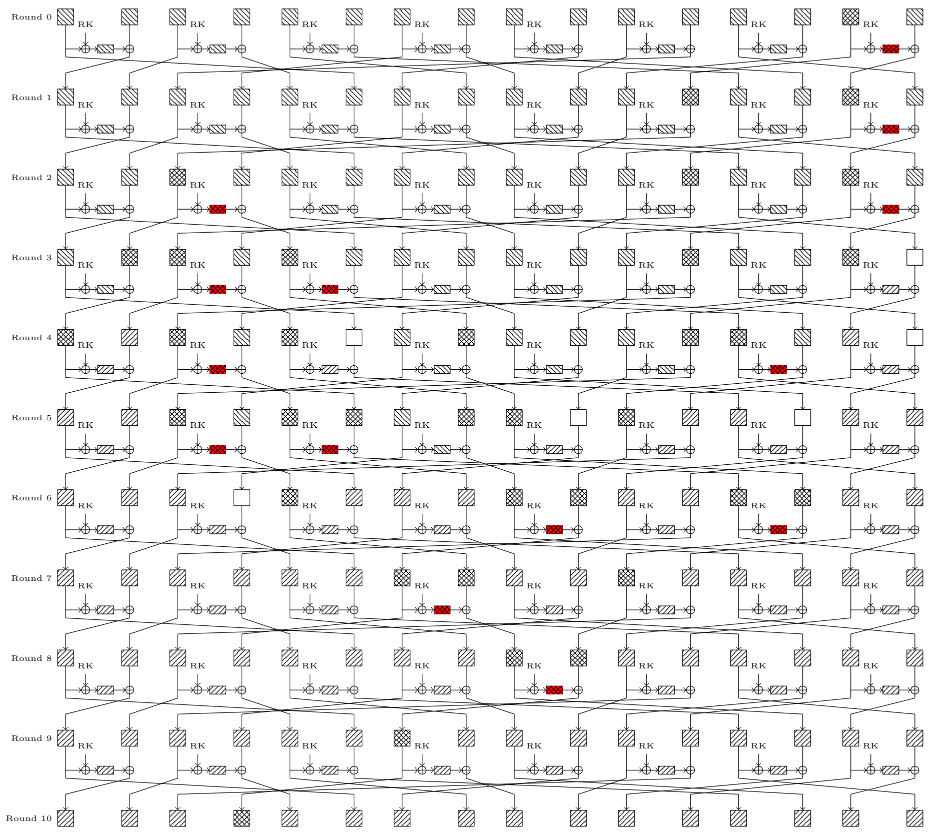


Fig. 21: Distinguisher for $\mathcal{DS}$-MITM attack on 20-round TWINE-80: A 10-round $\mathcal{DS}$-MITM distinguisher for TWINE-80 with $\mathcal{A} = [14]$ (the nibble denoted by crosshatch in round 0), $\mathcal{B} = [3]$ (the nibble denoted by crosshatch in round 10), and $\text{Deg}(\mathcal{A}, \mathcal{B}) = 14$ (the nibbles marked in red).

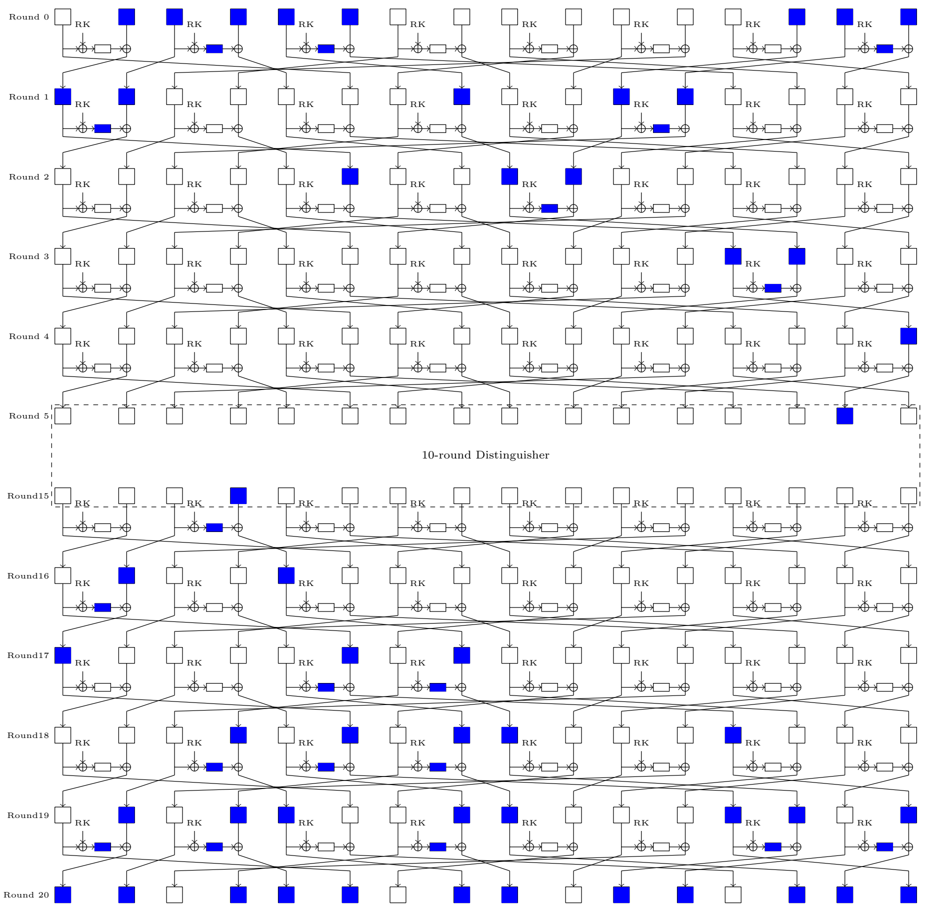


Fig. 22: Key recovery for $\mathcal{DS}$-MITM attack on 20-round `TWINE`-80: Backward differential and forward determination relationship in the outer rounds based on the 10-round distinguisher in Figure 21 as E_1 in the middle. Nibbles in Guess(E_0) and Guess(E_2) are marked in blue. From the input state in round 0, we know that $\bar{\mathcal{A}} = [1, 2, 3, 6, 7, 13, 14, 15]$, and hence the data complexity of the attack is $2^{8\times 4} = 2^{32}$.

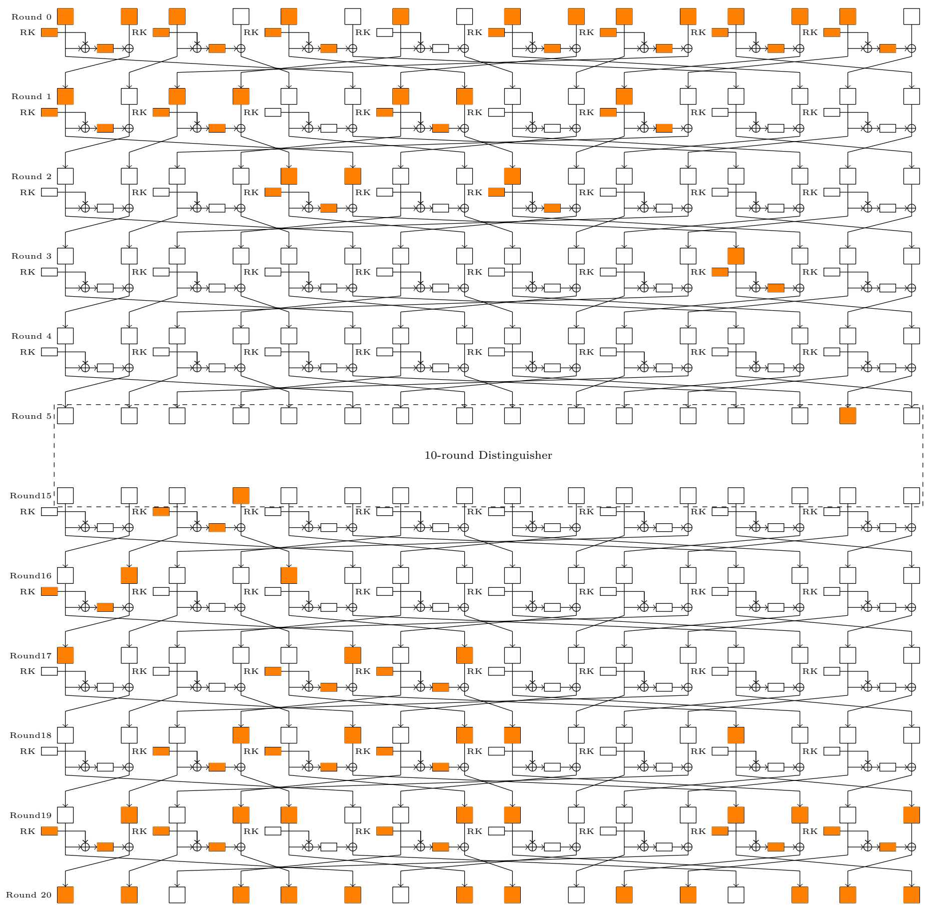


Fig. 23: Guessed values in $\mathcal{DS}$-MITM attack on 20-round `TWINE`-80: Derive k_{E_0} and k_{E_2} from Figure 22. The sub-keys marked in orange must be guessed (without key-bridging). With the knowledge of these nibbles, we can derive the values of those nibbles of P^0 marked in orange, from which we can determine all blue nibbles in Figure 22.

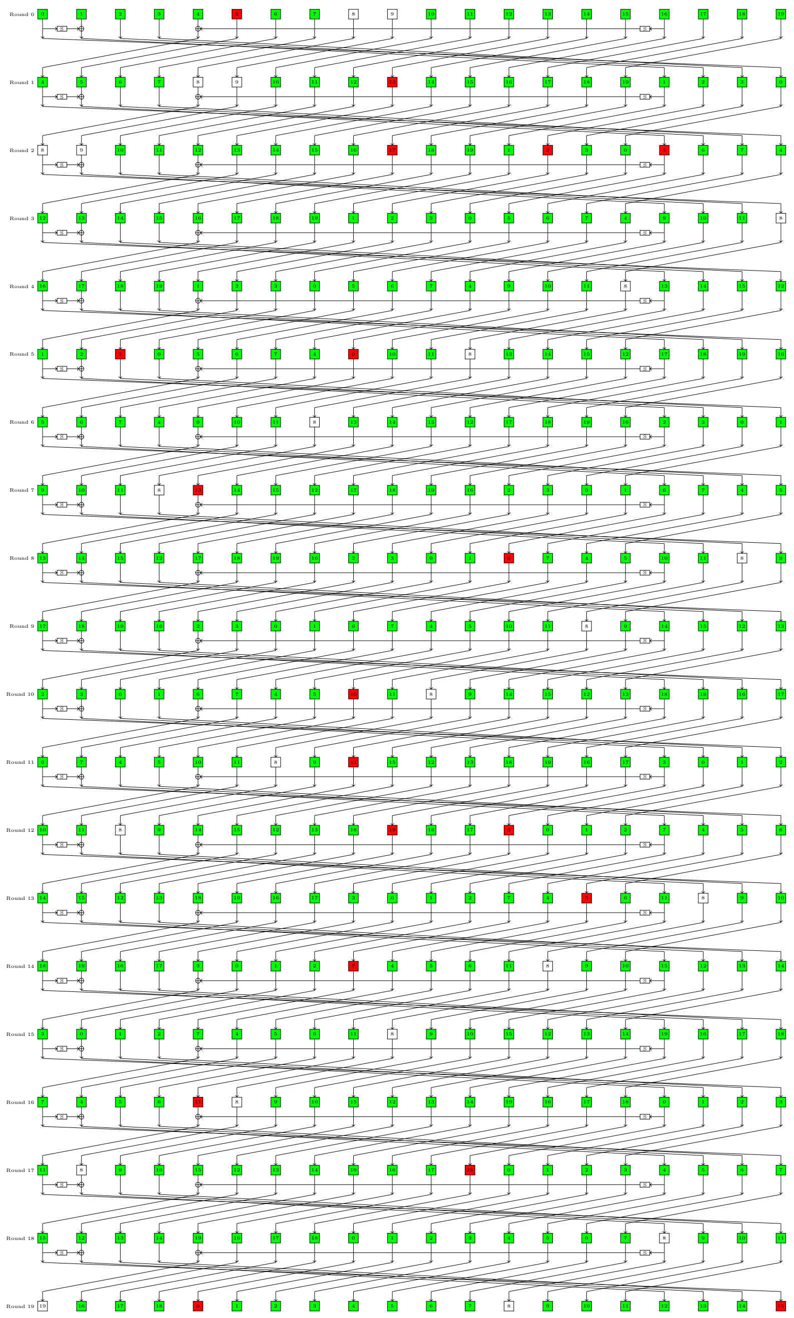


Fig. 24: Key-bridging in $\mathcal{DS}$-MITM attack on 20-round TWINE-80: All green sub-key nibbles can be uniquely deduced from the 19 sub-key nibbles marked in red. All sub-key nibbles corresponding to sub-keys colored in orange in Figure 23 are included in the guessed or determined sub-keys. In each round, sub-key nibbles $[1, 3, 4, 6, 13, 14, 15, 16]$ are used as round keys.

J $\mathcal{DS}$-MITM attack on 19-round SKINNY-128-256

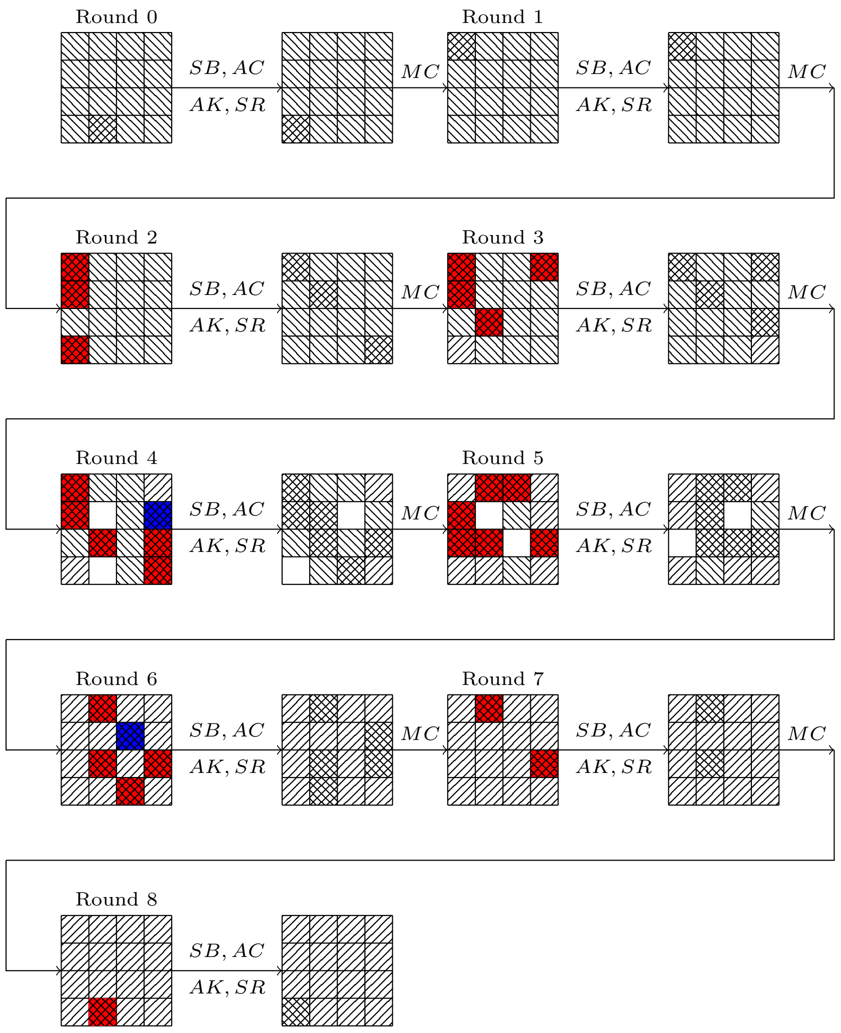


Fig. 25: Distinguisher for $\mathcal{DS}$-MITM attack on 19-round SKINNY-128-256: A 8.5-round $\mathcal{DS}$-MITM distinguisher for SKINNY-128-256 such that $\mathcal{A} = [13]$ (the cell denoted by crosshatch in round 0), $\mathcal{B} = [12]$ (the cell denoted by crosshatch in the last state array), and $\text{Deg}(\mathcal{A}, \mathcal{B}) = 25$ (the cells marked in red). The cells marked in blue can be determined from the red cells due to the connection relations imposed by the linear layer of SKINNY.

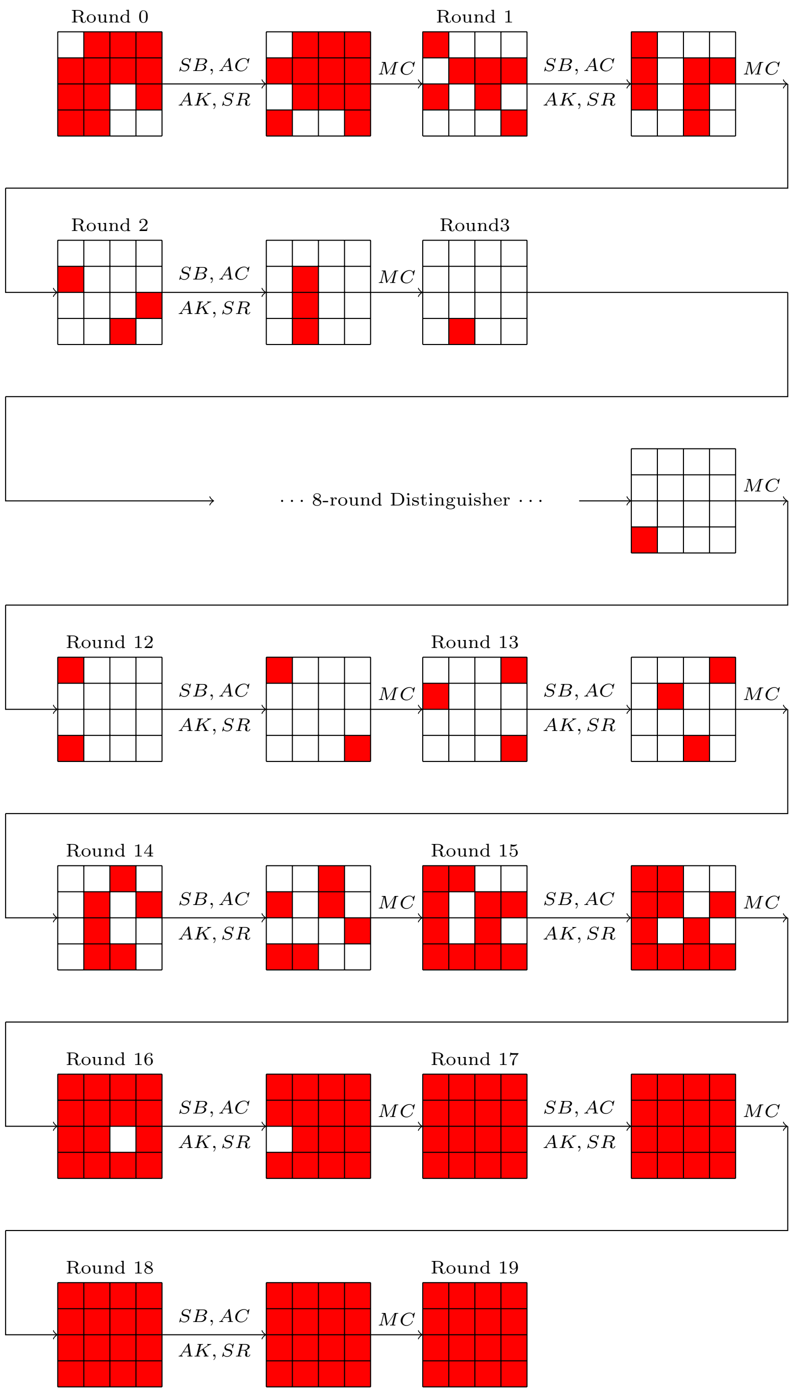


Fig. 26: Key recovery for $\mathcal{DS}$-MITM attack on 19-round `SKINNY`-128-256: Backward differential and forward determination relationship in the outer rounds based on the 8.5-round distinguisher in Figure 25 as E_1 in the middle. Cells in Guess(E_0) and Guess(E_2) are marked in red.

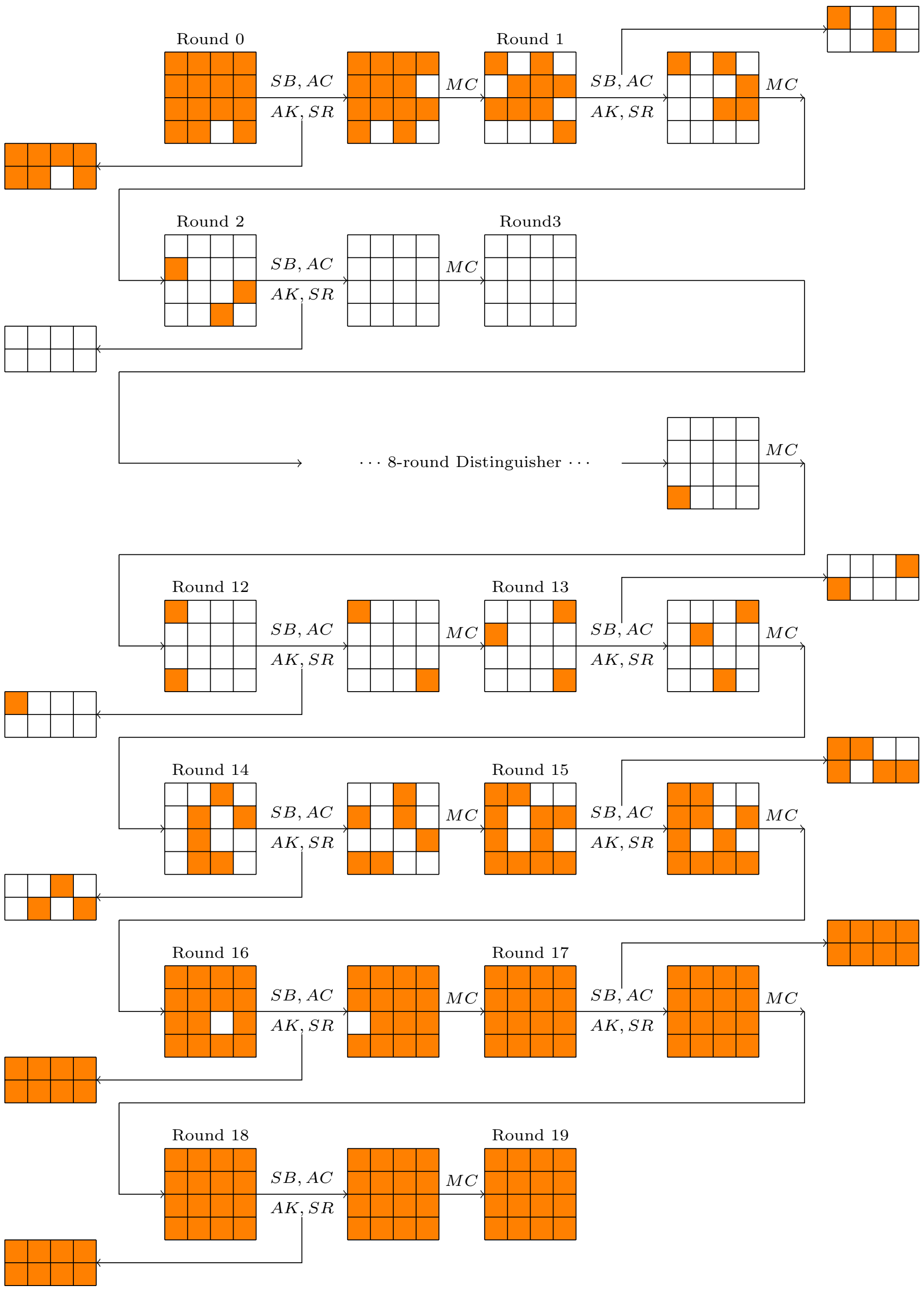


Fig. 27: Guessed values for $\mathcal{DS}$-MITM attack on 19-round `SKINNY`-128-256: Derive k_{E_0} and k_{E_2} from Figure 26. The sub-keys marked in orange must be guessed (without key-bridging). With the knowledge of these nibbles, we can derive the values of those nibbles of P^0 marked in orange, from which we can determine all red nibbles in Figure 26.

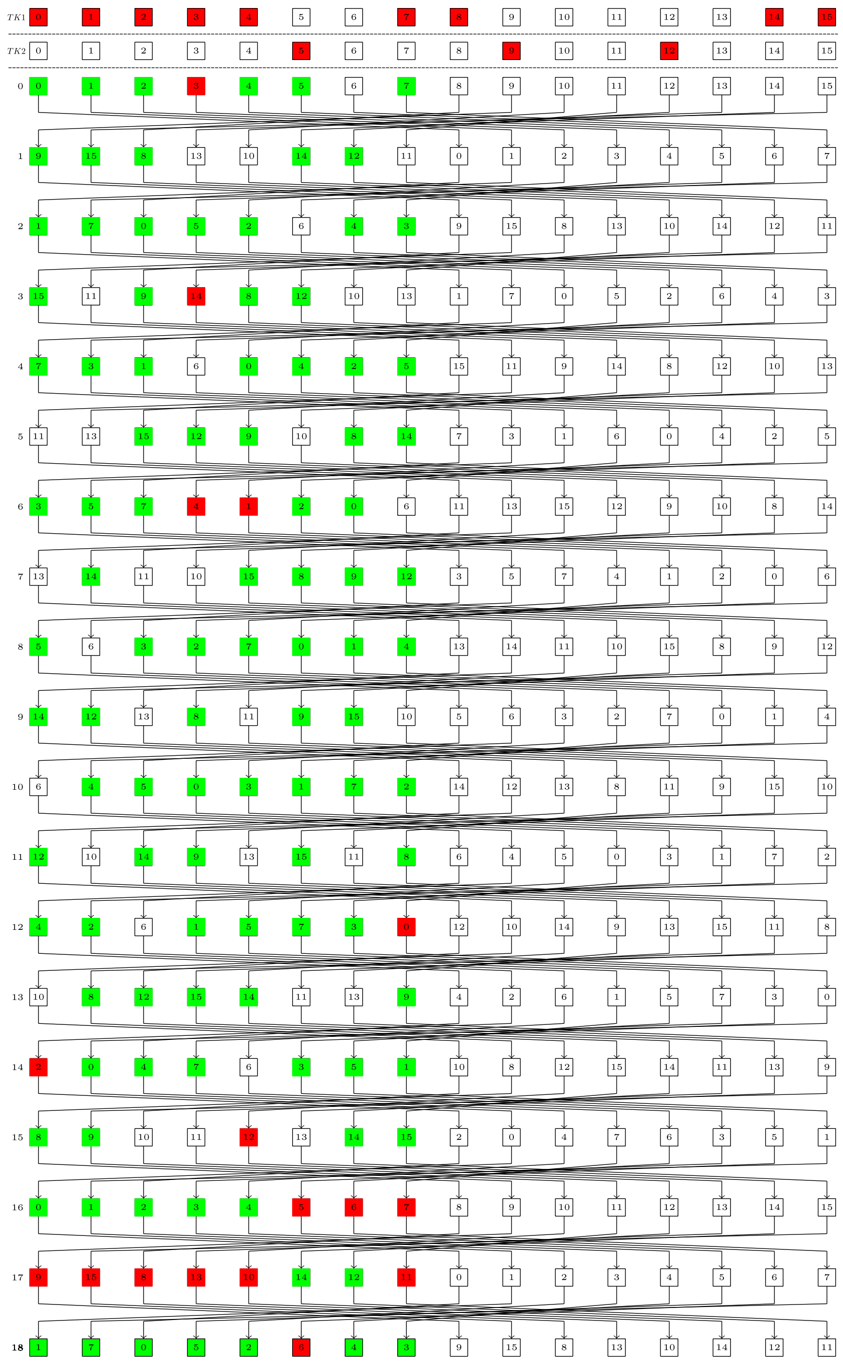


Fig. 28: Key-bridging for $\mathcal{DS}$-MITM attack on 19-round `SKINNY`-128-256: All green sub-key cells can be uniquely deduced from the sub-key nibbles marked in red. All sub-key cells corresponding to orange sub-keys in Figure 27 are included in the guessed or determined sub-keys.

K $\mathcal{DS}$-MITM attack on 21-round SKINNY-64-192

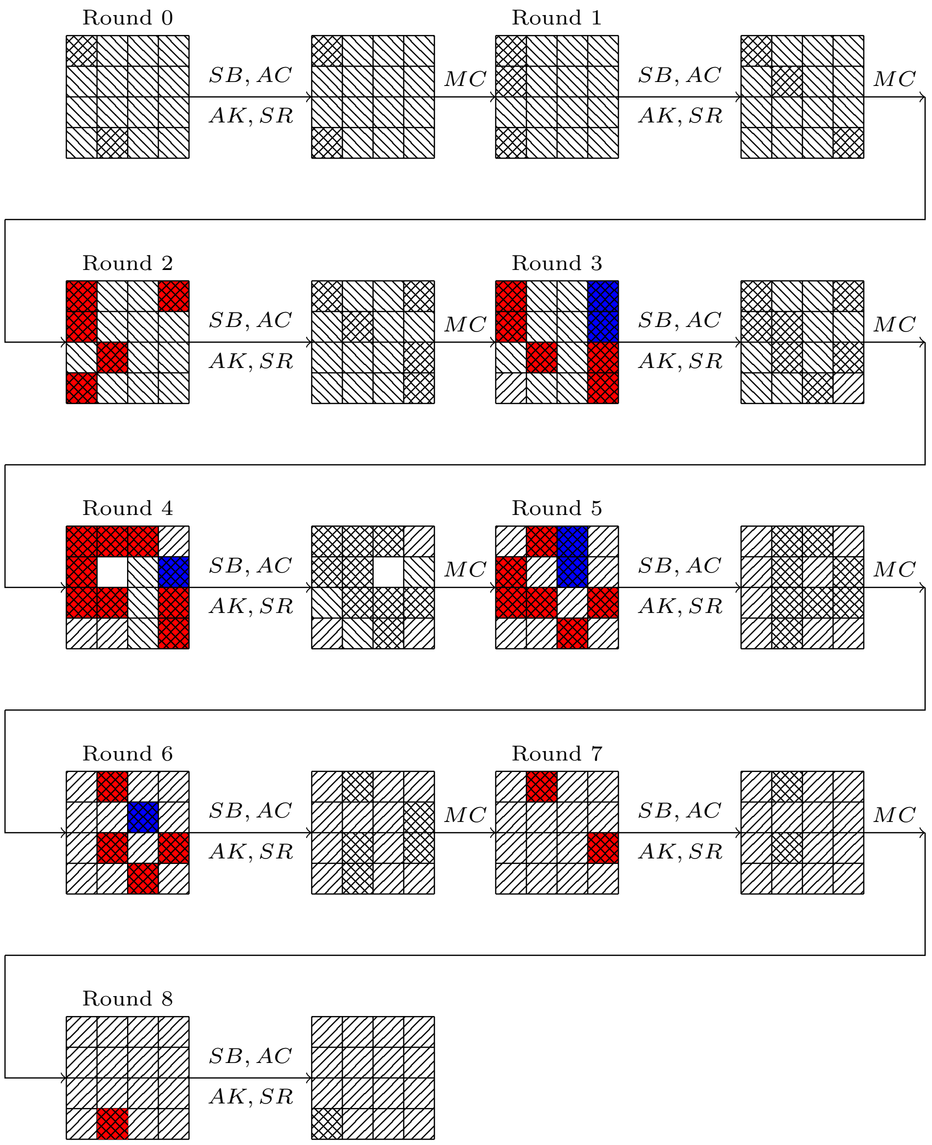


Fig. 29: Distinguisher for $\mathcal{DS}$-MITM attack on 21-round SKINNY-64-192: A 8.5-round $\mathcal{DS}$-MITM distinguisher for SKINNY-64-192 such that $\mathcal{A} = [0, 13]$ (the cell denoted by crosshatch in round 0), $\mathcal{B} = [12]$ (the cell denoted by crosshatch in the last state array), and $\mathrm{Deg}(\mathcal{A}, \mathcal{B}) = 31$ (the cells marked in red). The cells marked in blue can be determined from the red cells due to the connection relations imposed by the linear layer of SKINNY.

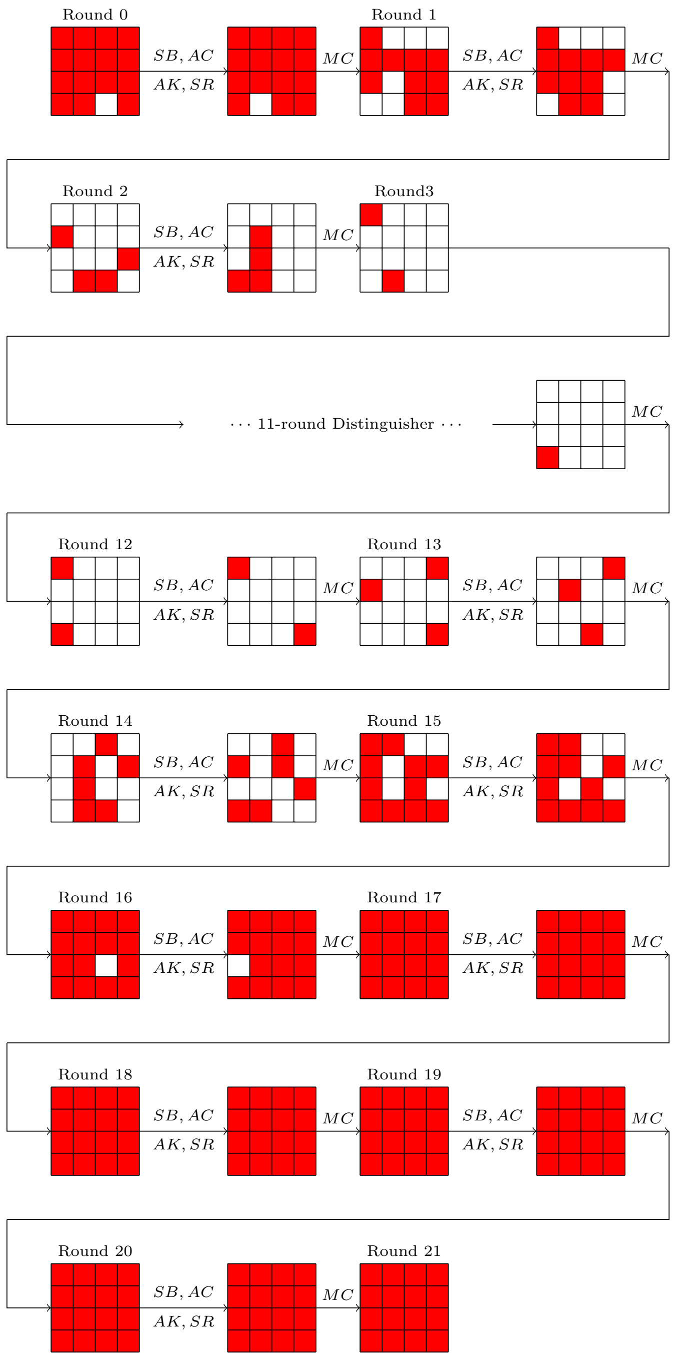


Fig. 30: Key recovery for $\mathcal{DS}$-MITM attack on 21-round SKINNY-64-192: Backward differential and forward determination relationship in the outer rounds based on the 8.5-round distinguisher in Figure 29 as E_1 in the middle. Cells in Guess(E_0) and Guess(E_2) are marked in red.

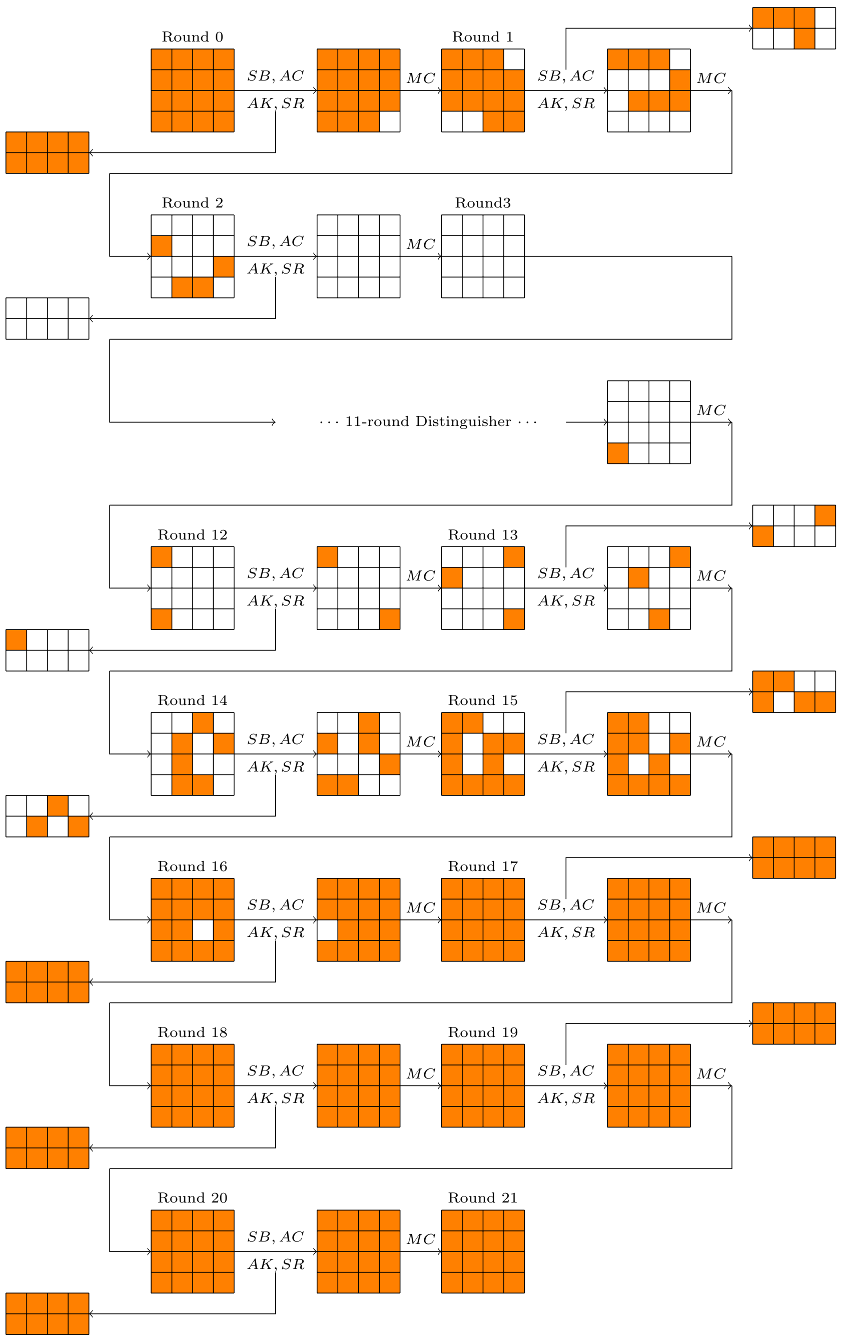


Fig. 31: Guessed values for $\mathcal{DS}$-MITM attack on 21-round `SKINNY`-64-192: Derive k_{E_0} and k_{E_2} from Figure 30. The sub-keys marked in orange must be guessed (without key-bridging). With the knowledge of these nibbles, we can derive the values of those nibbles of P^0 marked in orange, from which we can determine all red nibbles in Figure 30.

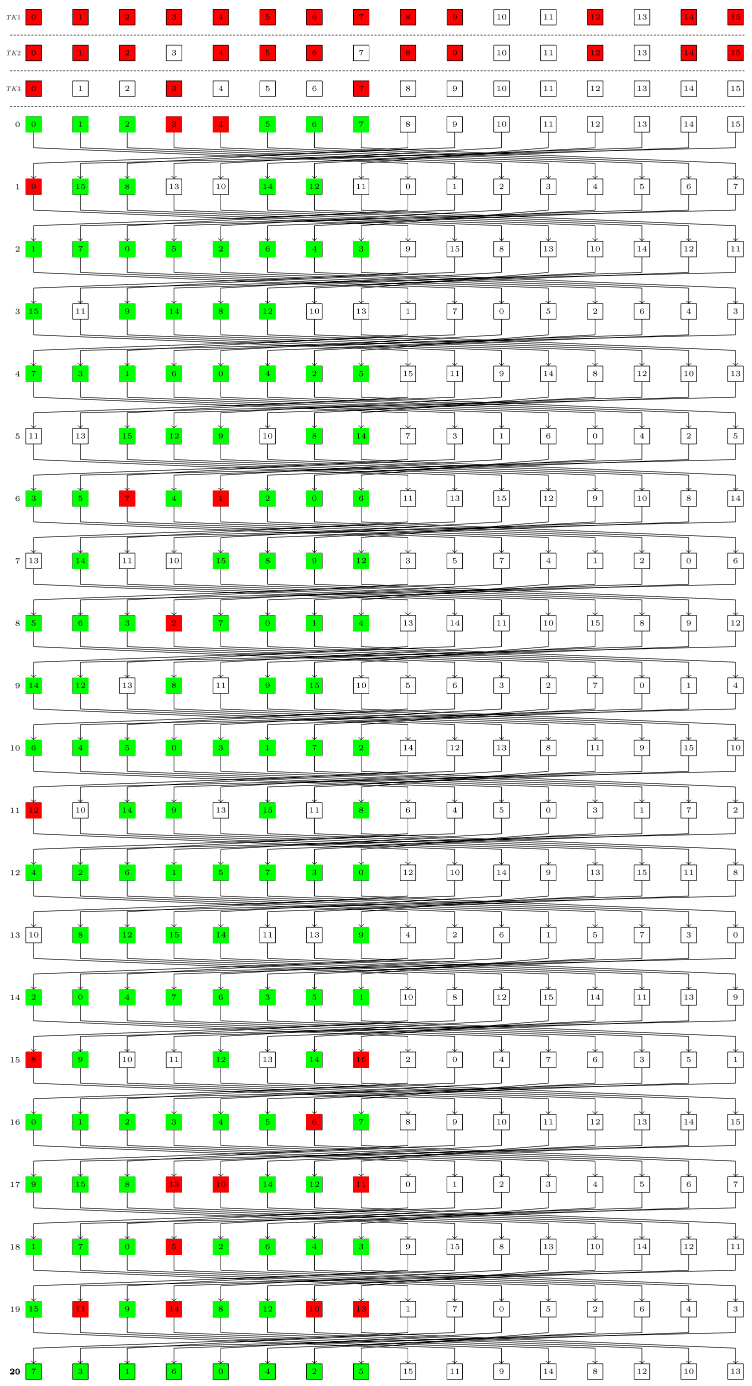


Fig. 32: Key-bridging for $\mathcal{DS}$-MITM attack on 21-round SKINNY-64-192: All green sub-key cells can be uniquely deduced from the sub-key nibbles marked in red. All sub-key cells corresponding to orange sub-keys in Figure 31 are included in the guessed or determined sub-keys.

L $\mathcal{DS}$-MITM attack on 18-round SKINNY-64-128

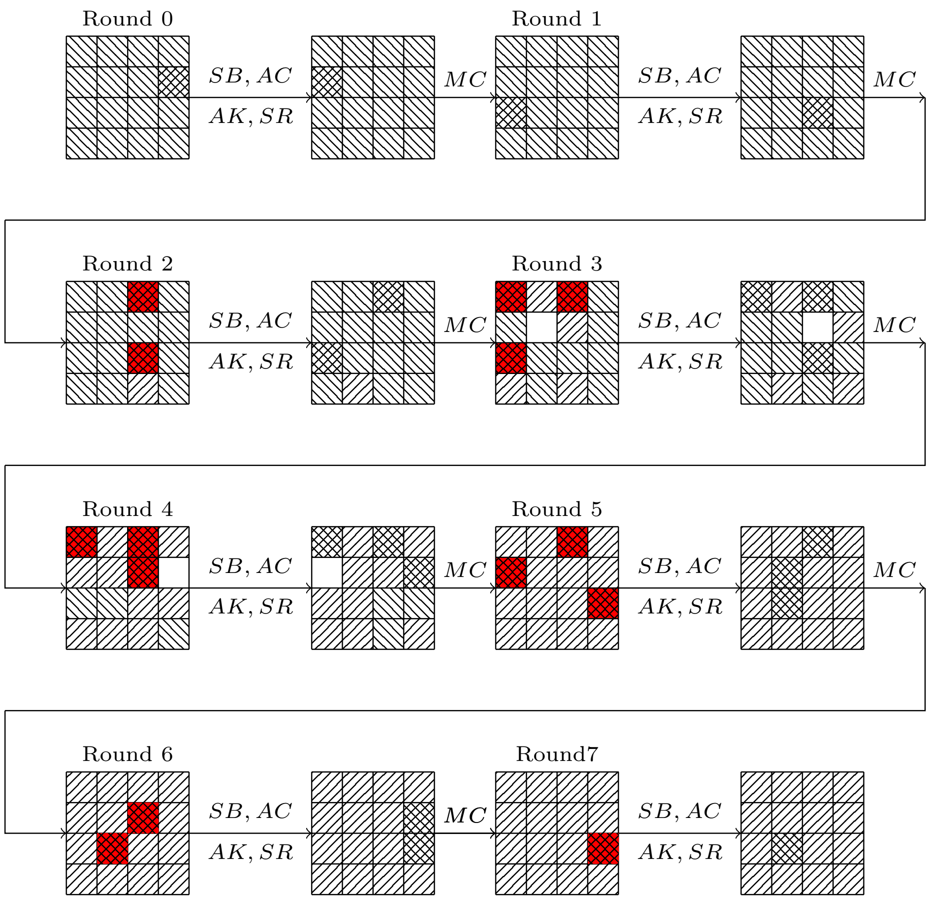


Fig. 33: Distinguisher for $\mathcal{DS}$-MITM attack on 18-round SKINNY-64-128: A 7.5-round $\mathcal{DS}$-MITM distinguisher for SKINNY-64-128 such that $\mathcal{A} = [7]$ (the cell denoted by crosshatch in round 0), $\mathcal{B} = [9]$ (the cell denoted by crosshatch in the last state array), and $\text{Deg}(\mathcal{A}, \mathcal{B}) = 14$ (the cells marked in red). The cells marked in blue can be determined from the red cells due to the connection relations imposed by the linear layer of SKINNY.

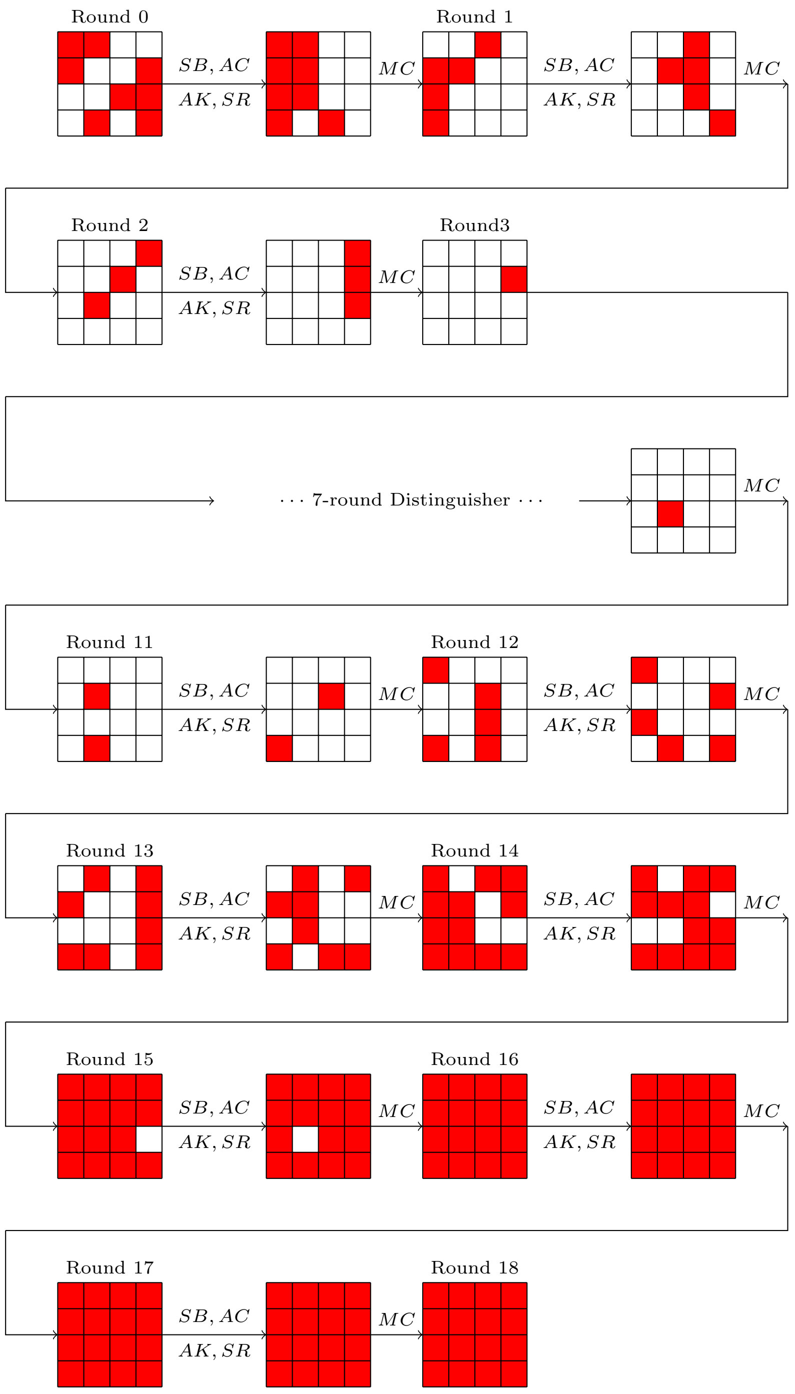


Fig. 34: Key recovery for $\mathcal{DS}$-MITM attack on 18-round `SKINNY`-64-128: Backward differential and forward determination relationship in the outer rounds based on the 7.5-round distinguisher in Figure 33 as E_1 in the middle. Cells in Guess(E_0) and Guess(E_2) are marked in red.

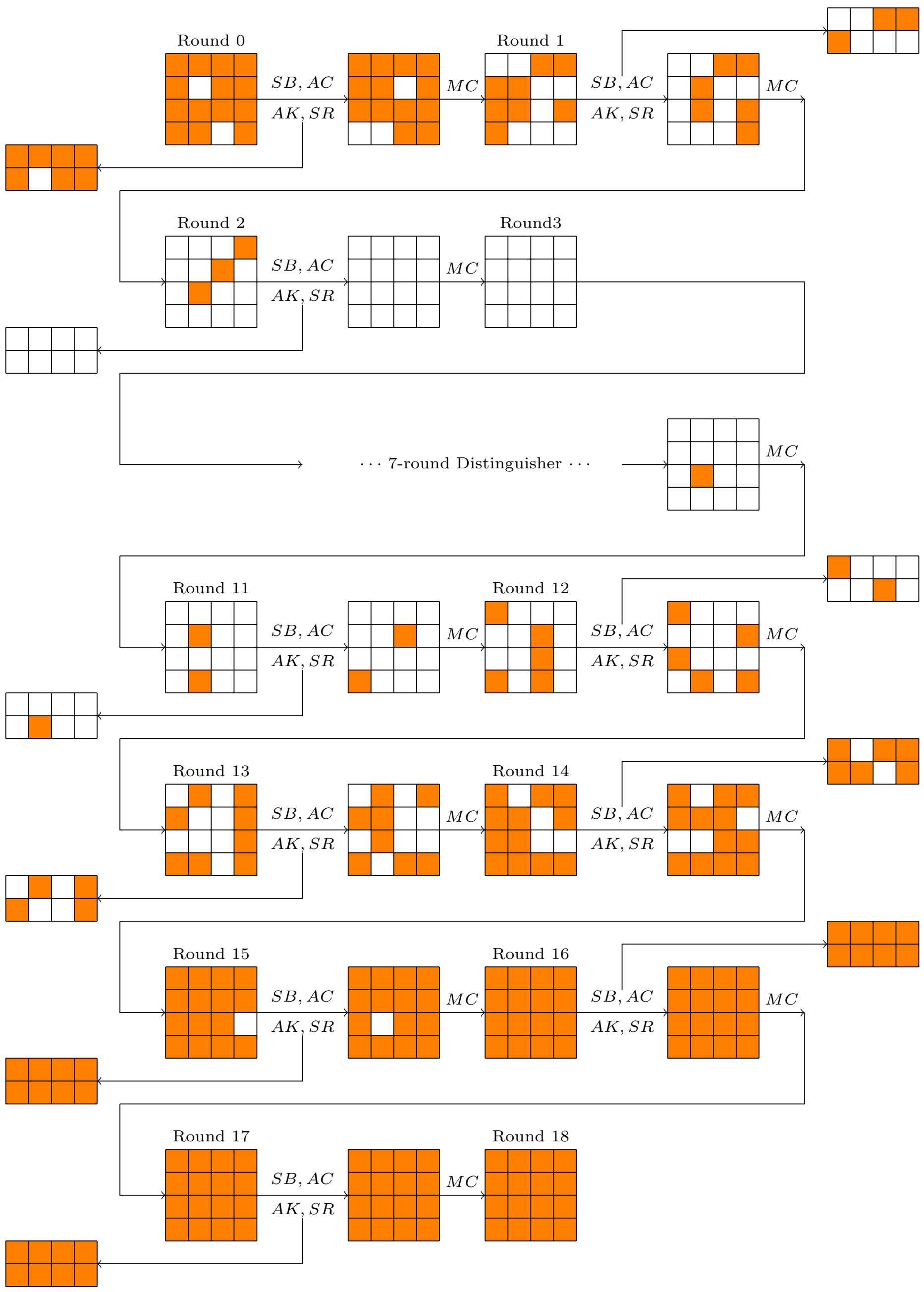


Fig. 35: Guessed values for $\mathcal{DS}$-MITM attack on 18-round `SKINNY`-64-128: Derive k_{E_0} and k_{E_2} from Figure 30. The sub-keys marked in orange must be guessed (without key-bridging). With the knowledge of these nibbles, we can derive the values of those nibbles of P^0 marked in orange, from which we can determine all red nibbles in Figure 34.

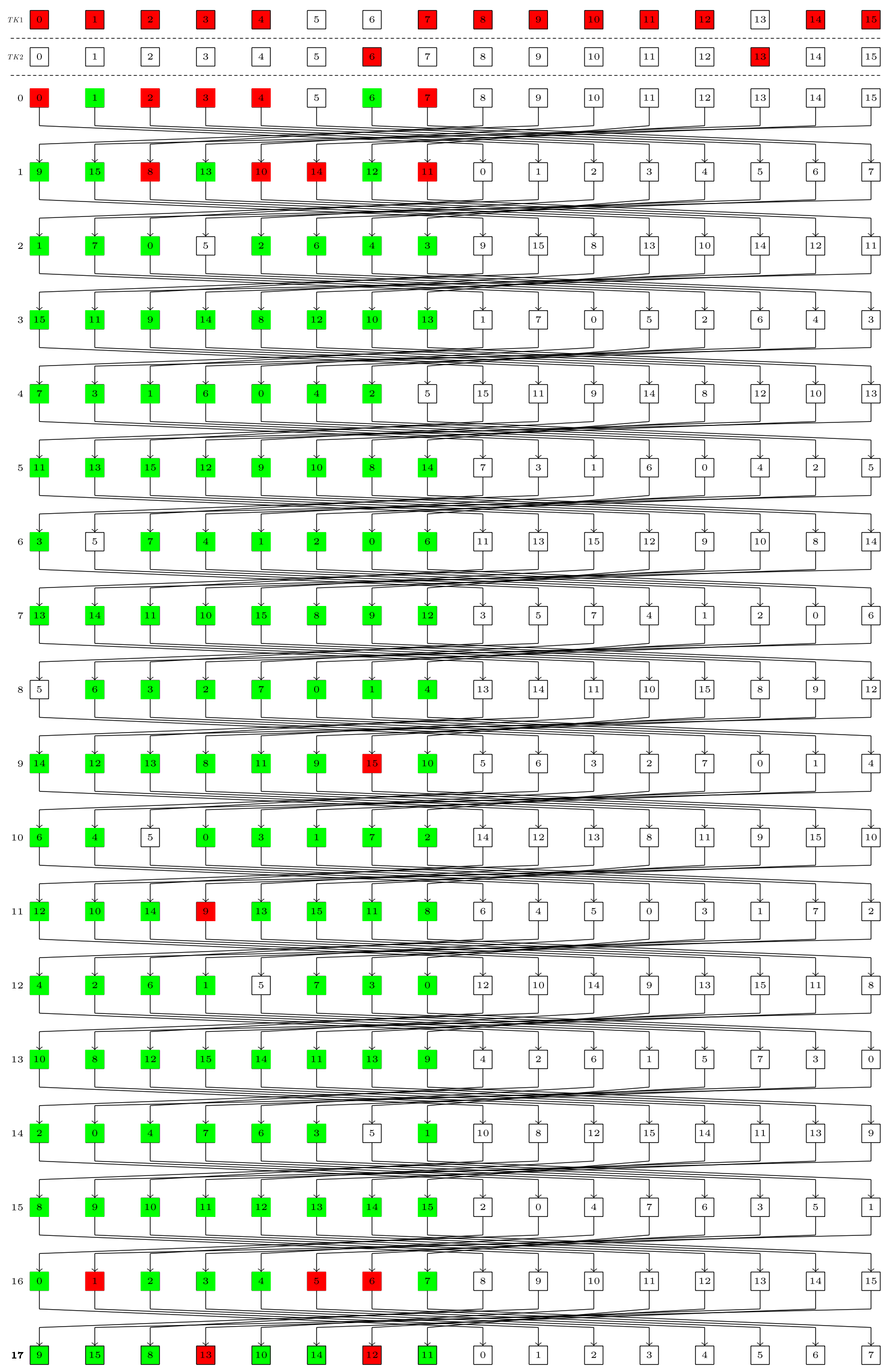


Fig. 36: Key-bridging for $\mathcal{DS}$-MITM attack on 18-round `SKINNY-64-128`: All green sub-key cells can be uniquely deduced from the sub-key nibbles marked in red. All sub-key cells corresponding to orange sub-keys in Figure 35 are included in the guessed or determined sub-keys.