Speeding up Preimage and Key-Recovery Attacks with Highly Biased Differential-Linear Approximations

Zhongfeng Niu[1], Kai Hu[2,4,6], Siwei Sun[1,3]⋆, Zhiyu Zhang[1], Meiqin Wang[2,4,5]

[1] School of Cryptology, University of Chinese Academy of Sciences, Beijing, China {niuzhongfeng, sunsiwei}@ucas.ac.cn zhangzhiyu14@mails.ucas.ac.cn

[2] School of Cyber Science and Technology, Shandong University, Qingdao, Shandong, China {kai.hu, mqwang}@sdu.edu.cn

[3] State Key Laboratory of Cryptology, P.O. Box 5159, Beijing 100878, China

[4] Key Laboratory of Cryptologic Technology and Information Security, Ministry of Education, Shandong University, Jinan, China.

[5] Quan Cheng Shandong Laboratory, Jinan, China

[6] School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore, Singapore

Abstract. We present a framework for speeding up the search for preimages of candidate one-way functions based on highly biased differential-linear distinguishers. It is naturally applicable to preimage attacks on hash functions. Further, a variant of this framework applied to keyed functions leads to accelerated key-recovery attacks. Interestingly, our technique is able to exploit *related-key* differential-linear distinguishers in the *single-key* model without querying the target encryption oracle with unknown but related keys. This is in essence similar to how we speed up the key search based on the well known complementation property of DES, which calls for caution from the designers in building primitives meant to be secure in the single-key setting without a thorough cryptanalysis in the related-key model. We apply the method to sponge-based hash function `Ascon-HASH`, XOFs `XOEsch`/`Ascon-XOF` and AEAD `Schwaemm`, etc. Accelerated preimage or key-recovery attacks are obtained. Note that all the differential-linear distinguishers employed in this work are highly biased and thus can be experimentally verified.

Keywords: Differential-linear, Preimage attack, Key-recovery attack, Sponge function, Hash function, AEAD

1 Introduction

Searching for preimages and secret keys are central topics in the cryptanalysis of symmetric-key cryptographic primitives. However, we see obvious limitations of the currently available techniques when it comes to the cryptanalysis

⋆ The corresponding author

of permutation-based primitives, especially the sponge-based constructions [BDPVA07, BDPA08], which underlies the design of the SHA3 standard [BDPA13] and the new NIST LWC standard `Ascon` [DEMS21]. Since the sponge construction has emerged as a versatile tool for building various cryptographic primitives, including hash functions [BDP+18], Message Authentication Codes [BDPA11], and Authenticated Encryption with Associated Data (AEAD) schemes [DEMS21], cryptanalysis of sponge-based constructions has drawn a lot of attention.

The linear structure [GLS16, LSLW17, LS19, HLY21, LIMY21] and Meet-in-the-Middle (MitM) techniques [QHD+23, QZH+23] are two major strategies for preimage attacks on sponge-based hash functions, whereas the cube-like attacks [DMP+15, DEMS15, HWX+17, BCP22] have dominated the key-recovery attacks on sponge-based AEADs. However, for ciphers with more complicated round functions (e.g., ARX constructions), the applicability of the above-mentioned techniques are extremely limited. For example, to the best of our knowledge, there are no preimage attacks on `Esch`/`XOEsch` (the hash function/XOF in the NIST LWC finalist `Sparkle` suite) from the open literature.[1] In terms of its AEAD `Schwaemm`, all known attacks are proposed by its designers, which either require a data complexity beyond the data limit imposed by the designers or omit the whitening operation, making the attacks invalid. As summarized by the NIST report:

> "*All of these attacks on* `Schwaemm` *variants require data beyond the data limit made by the submitters ... There is no known cryptanalysis on the hash variants ...*" (see [TMC+23, Page 34])

Despite this situation, it seems that we are more capable of performing distinguishing attacks on the underlying permutations of the sponge-based constructions. In particular, the differential-linear (DL) technique [LH94] is frequently employed and often shows remarkable effectiveness against permutation-based primitives, including ARX designs [BBC+22]. In [NSLL22], several DL distinguishers for the round-reduced `Sparkle` permutation (the underlying permutation of `Schwaemm`) or its building blocks were identified. There exist even *deterministic* DL approximations for the 4-round `Alzette` [BBdS+20], the core non-linear component of `Sparkle`. For the `Ascon` permutation, 4- and 5-round practical DL distinguishers were found [DEMS15, BDKW19], much stronger than differential or linear distinguishers. Nevertheless, it is challenging to translate these distinguishers into meaningful (preimage or key-recovery) attacks.

Our Contributions. We propose a strategy for speeding up the search for a preimage of a one-way function based on highly biased differential-linear distinguishers.[2] By regarding a keyed primitive as a parameterized one-way function, the strategy can be adapted to accelerate key-recovery attacks. We demonstrate

[1] In [SS22], Schrottenloher and Stevens identified practical distinguishers of the `Sparkle` permutation with the MitM technique. However, there is no obvious way to transform them into meaningful attacks on the corresponding hash or AEAD mode.

[2] Strictly, the one-way function here should be called a candidate for one-way function, since for ideal one-way functions there should be no highly biased distinguishers.

the versatility of the method by applying it to various cryptographic primitives with sponge constructions, small S-boxes, or ARX components. The results are summarized in Table 1 and Table 2.

Preimage Attacks on `XOEsch` *and* `Ascon-XOF`. We present the first preimage attacks on `XOEsch`384 and `XOEsch`256 reduced to 1.5 and 2.5 steps (0.5 step means an extra nonlinear layer). For `Ascon-XOF`, our method provides a new approach for performing preimage attacks which can reach 4 rounds of `Ascon-XOF` as the linearization [LHC+23] and MitM techniques [QHD+23]. All of our attacks require negligible or insignificant memory.

Key-Recovery Attacks on `Schwaemm`. When there are a huge number of highly biased differential-linear approximations, we introduce a dedicated time-memory trade-off technique relying on a large hash table for testing the equations induced by the differential-linear approximations, based on which we present the first valid key-recovery attacks on the AEAD `Schwaemm`. For 3.5 and 4.5 steps of `Schwaemm`, our attacks are about 2^{63} or 2^{126} times faster than the *brute-force search*. These attacks bear some resemblances to Daemen's attack on Even-Mansour [Dae91] and the "slidex" attack [DKS12] on `Prince` [BCG+12].

Key-Recovery Attacks on Full `Crax-S`*-10*. We present a key-recovery attack on the full `Crax-S-10` in the *single-key* model by exploiting a set of 1-round *related-key* DL distinguishers, and it is $2^{0.47}$ times faster than the brute-force key search. This attack does not have any practical impact on the security of `Crax-S-10`. However, it calls for caution from the designers in designing primitives meant to be secure in the single-key setting without a thorough cryptanalysis in the related-key model. Note that there are other attacks able to exploit related-key distinguishers in the single-key model (e.g., the biclique attacks [BKR11]).

New Preimage Attacks on Sponge-based Hash Functions. We propose a new framework for preimage attacks on sponge-based hash functions, which is useful for a hash function claiming a security level higher than half of its capacity. The framework first recovers a capacity part in the squeezing phase, then uses an internal-collision phase to meet the initial input specified by the targeted hash function. With Floyd's cycle-finding algorithm [Flo67, Sas14], the inner-collision phase requires a negligible amount of memory. At CRYPTO 2022 [LM22], Lefevre and Mennink proved that the preimage security bound of a sponge-based hash built on a random permutation is $\min\{\max\{2^{n-r'}, 2^{c/2}\}, 2^n\}$ work, where n is the digest size, c is the capacity of the sponge (during absorption), and r' is the rate (during squeezing). As a result, the security bound of `Ascon-HASH` against preimage attacks can be improved to 192-bit. Under the new security bound, we manage to give a preimage attack on up to 4 rounds of `Ascon-HASH`. However, we note that our work does not influence the original design since the designers only claim a 128-bit security.

Limitations. Our method is an exhaustive search in nature, similar to the technique employed to speed up the exhaustive key search based on the complementation property of DES [HMS+76]. When the number of highly biased differential-linear distinguishers is small, the speed-up effect is marginal. When there are a huge number of differential-linear distinguishers, we can skip a lot of evaluations of the targeted one-way function during the exhaustive search. However, in this case, the complexity for testing the differential-linear approximation equations is not negligible and thus we have to use large hash tables to avoid the corresponding complexities. Moreover, our method is only effective with highly biased differential-linear approximations, which is difficult to find in general.

Table 1: The preimage and collision attacks on `XOEsch`, `Ascon-XOF` and `Ascon-HASH`. Except for the 6-round preimage attack on `Ascon-XOF`, the success probability of all preimage attacks in this table are approximately 0.63.

Target	Attack type	Round (Step)	Time	Mem.	Output length	Security claim	Meth.	Ref.
`XOEsch384`	Preimage	1.5	$2^{123.64}$	Neg.	128	2^{128}	DL	Sect. 5.2
			$2^{186.64}$	Neg.	192	2^{192}	DL	Sect. 5.2
		2.5	$2^{125.76}$	2^{11}	128	2^{128}	DL	Sect. 5.3
			$2^{188.76}$	2^{11}	192	2^{192}	DL	Sect. 5.3
`XOEsch256`	Preimage	1.5	$2^{123.64}$	Neg.	128	2^{128}	DL	Sect. F.1
		2.5	$2^{125.66}$	2^{11}	128	2^{128}	DL	Sect. F.2
`Ascon-XOF`	Preimage	2	2^{103}	Neg.	128	2^{128}	Cube-like	[ASC]
		3	$2^{120.58}$	2^{39}	128	2^{128}	MitM	[QHD+23]
		3	$2^{114.53}$	2^{30}	128	2^{128}	MitM	[QZH+23]
		3	$2^{112.21}$	Neg.	128	2^{128}	Lin.	[LHC+23]
		3	$2^{120.02}$	Neg.	128	2^{128}	DL	Sect. K
		4	$2^{124.67}$	2^{50}	128	2^{128}	MitM	[QHD+23]
		4	$2^{124.49}$	Neg.	128	2^{128}	Lin.	[LHC+23]
		4	$2^{125.47}$	Neg.	128	2^{128}	DL	Sect. 6
		6†	$2^{127.3}$	Neg.	128	2^{128}	Algebraic	[DEMS21]
`Ascon-HASH`	Preimage	3	$2^{183.98}$	Neg.	256	2^{192}	MitM-DL	Sect. L
		4	$2^{188.61}$	Neg.	256	2^{192}	MitM-DL	Sect. 7
	Collision	2	2^{125}	Neg.	128	2^{128}	Diff.	[ZDW19]
		2	2^{103}	Neg.	128	2^{128}	Diff.	[GPT21]
		3	$2^{121.85}$	2^{121}	128	2^{128}	MitM	[QZH+23]
		4	$2^{126.77}$	2^{126}	128	2^{128}	MitM	[QZH+23]

DL: Differential-linear, Lin.: Linearization, †: No padding bits

2 Notations and Preliminaries

For a positive integer a, we denote by $[a]$ the set $\{0, 1, \ldots, a-1\}$, $\log(a)$ the base-2 logarithm of a and $\ln(a)$ the base-e logarithm of a. Let $\mathbb{F}_2 = \{0, 1\}$ be the binary field and $\mathbb{F}_2^n$ be the set of all n-bit strings. For $x \in \mathbb{F}_2^n$, $wt(x)$ represents the Hamming weight of x. The exclusive-or of $x \in \mathbb{F}_2^n$ and $y \in \mathbb{F}_2^n$ is denoted by $x \oplus y$, and $x \cdot y = \bigoplus_{i=0}^{n-1} x_i y_i$ is the dot product, where x_i and y_i are the i-th bits of x and y, respectively. For $x \in \mathbb{F}_2^n$, and $\mathbb{A} \subseteq \mathbb{F}_2^n$, we overload the $\oplus$ operator and define the x-translation $x \oplus \mathbb{A}$ of $\mathbb{A}$ to be the set $\{x \oplus y : y \in \mathbb{A}\}$. The set $\mathbb{A} \cup \{0\}$

Table 2: Results on AEADs and block ciphers. Note that all previous state-recovery attacks on `Schwaemm` AEADs either omit the whitening (labeled by $\ominus$) or surpass the data limit set by designers (labeled by $\oslash$). The success probability of all our key-recovery attacks for 4.5-step `Schwaemm` is 0.63.

Target	Attack type	Step	Time	Data	Mem.	Security claim	Method	Ref.
`Schwaemm` 256-128	Key-rec.	3.5	$2^{65.3}$	2^{64}	2^{64}	2^{120}	DL	Sect. 8.1
		3.5	2^{64}	1	Neg.	2^{120}	Structural	Sect. M
		4.5	$2^{65.4}$	2^{64}	2^{64}	2^{120}	DL	Sect. 8.2
`Schwaemm` 192-192	State-rec.$\ominus$	3.5	2^{128}	2^{64}	2^{128}	2^{184}	Data T-O	[BBdS+21]
	Key-rec.	3.5	2^{129}	2^{64}	2^{64}	2^{184}	DL	Sect. N.1
	State-rec.$\ominus$	4.5	$2^{128+\tau}$	$2^{128-\tau}$	$2^{128+\tau}$	2^{184}	Bir. Diff.	[BBdS+21]
	Key-rec.	4.5	2^{129}	2^{64}	2^{64}	2^{184}	DL	Sect. N.1
`Schwaemm` 256-256	State-rec.$\ominus$	3.5	2^{192}	2^{64}	2^{192}	2^{248}	Data T-O	[BBdS+21]
	State-rec.$\ominus$	3.5	2^{192}	1	Neg.	2^{248}	Bir. Diff.	[BBdS+21]
	State-rec.$\oslash$	3.5	$2^{224+\tau}$	$2^{224-\tau}$	$2^{224+\tau}$	2^{248}	Bir. Diff.	[BBdS+21]
	Key-rec.	3.5	$2^{129.32}$	2^{128}	2^{128}	2^{248}	DL	Sect. N.2
	State-rec.$\ominus$	4.5	$2^{192}+2^{160+\tau}$	$2^{160-\tau}$	2^{192}	2^{248}	Bir. Diff.	[BBdS+21]
	Key-rec.	4.5	$2^{129.37}$	2^{128}	2^{128}	2^{248}	DL	Sect. N.2
`Schwaemm` 128-128	State-rec.$\ominus$	3.5	2^{64}	2^{64}	2^{64}	2^{120}	Data T-O	[BBdS+21]
	Key-rec.	3.5	$2^{65.32}$	2^{64}	2^{64}	2^{120}	DL	Sect. N.3
	State-rec.$\ominus$	4.5	$2^{96+\tau}$	$2^{96-\tau}$	$2^{96+\tau}$	2^{120}	Guess Det.	[BBdS+21]
	Key-rec.	4.5	$2^{65.37}$	2^{64}	2^{64}	2^{120}	DL	Sect. N.3
`Crax-S-10`	Key-rec.	10	$2^{127.53}$	2	Neg.	2^{128}	DL	Sect. O

DL: Differential-linear, Data. T-O: Data trade-off, Bir. Diff.: Birthday differential

is abbreviated as $\hat{\mathbb{A}}$, and $\langle \mathbb{A} \rangle$ represents the linear space spanned by $\mathbb{A}$. Thus $\langle \mathbb{A} \rangle = \langle \hat{\mathbb{A}} \rangle$. Let $\mathbb{V} \subseteq \mathbb{F}_2^n$ be a linear space of dimension $\dim(\mathbb{V}) = d$ spanned by $\{\alpha_0, \ldots, \alpha_{d-1}\}$. Then, let $\mathbb{V}^{\dashv} \subseteq \mathbb{F}_2^n$ be a linear space spanned by $\{\beta_0, \ldots, \beta_{n-d-1}\}$ such that the vectors in $\{\alpha_0, \ldots, \alpha_{d-1}, \beta_0, \ldots, \beta_{n-d-1}\}$ are linearly independent. Then, $\mathbb{F}_2^n$ can be split into a direct sum $\mathbb{V} \oplus \mathbb{V}^{\dashv}$, where $\mathbb{V}^{\dashv}$ contains $|\mathbb{V}^{\dashv}| = 2^{n-d}$ elements and $\dim(\mathbb{V}^{\dashv}) = n - d$. We call $\mathbb{V}^{\dashv}$ an algebraic complementary of $\mathbb{V}$.

Lemma 1. *Let $\mathbb{V} \subseteq \mathbb{F}_2^n$ be a linear space. Then $\bigcup_{x \in \mathbb{V}^{\dashv}} x \oplus \mathbb{V} = \mathbb{F}_2^n$. Moreover, For $x, y \in \mathbb{V}^{\dashv}$, $x \oplus \mathbb{V} \cap y \oplus \mathbb{V} \neq \emptyset$ if and only if $x = y$. That is, the $2^{n-\dim(\mathbb{V})}$ subsets $x \oplus \mathbb{V}$ with $x \in \mathbb{V}^{\dashv}$ form a partition of $\mathbb{F}_2^n$.*

Remark 1. For a linear space $\mathbb{V} \subseteq \mathbb{F}_2^n$, $\mathbb{V}^{\dashv}$ is not always equal to $\mathbb{V}^{\perp} = \{x \in \mathbb{F}_2^n : x \cdot y = 0 \text{ for all } y \in \mathbb{V}\}$. For example, if $\mathbb{V} = \{\texttt{00}, \texttt{11}\} \in \mathbb{F}_2^2$, then a choice of $\mathbb{V}^{\dashv}$ is $\{\texttt{00}, \texttt{01}\}$, and $\mathbb{V}^{\perp} = \{\texttt{00}, \texttt{11}\}$. But if $\mathbb{V}$ is spanned by unit vectors, then $\mathbb{V}^{\perp}$ is an algebraic complementary of $\mathbb{V}$. Note that may be other algebraic complementaries. For example, Let $\mathbb{V} = \langle \texttt{1000}, \texttt{0100} \rangle$. Then, both $\langle \texttt{1111}, \texttt{1010} \rangle$ and $\langle \texttt{0010}, \texttt{0001} \rangle$ are algebraic complementaries of $\mathbb{V}$.

Let $f : \mathbb{F}_2^m \to \mathbb{F}_2^n$ be a vectorial Boolean function. The correlation $\mathfrak{c}$ of the differential-linear approximation of f with input difference $\delta \in \mathbb{F}_2^m$ and linear mask $\lambda \in \mathbb{F}_2^n$ is defined as $\mathfrak{c} = \frac{1}{2^m} \sum_{x \in \mathbb{F}_2^m} (-1)^{\lambda \cdot (f(x) \oplus f(x \oplus \delta))}$, where $-1 \leq \mathfrak{c} \leq$

1 [LH94]. Note that when $\mathfrak{c} > 0$, the value of $\lambda \cdot (f(x) \oplus f(x \oplus \delta))$ is biased towards 0, and when $\mathfrak{c} < 0$, the value of $\lambda \cdot (f(x) \oplus f(x \oplus \delta))$ is biased towards 1. In short, we have $\Pr[\lambda \cdot (f(x) \oplus f(x \oplus \delta)) = 0] = \frac{1}{2} + \frac{\mathfrak{c}}{2}$, i.e., when $\mathfrak{c} \neq 0$, $\lambda \cdot (f(x) \oplus f(x \oplus \delta))$ is biased towards $\zeta_{\mathfrak{c}} = \frac{(-1)^{\mathtt{Sign}(\mathfrak{c})}+1}{2}$, where $\mathtt{Sign}(z) = 1$ when $z > 0$, and $\mathtt{Sign}(z) = 0$ when $z < 0$. One DL distinguisher of f with the difference-mask (δ, λ) whose correlation is $\mathfrak{c}$ is denoted by $\delta \xrightarrow[\mathfrak{c}]{f} \lambda$.

3 Speed up Preimage Recovery with DL Distinguishers

Let $F : \mathbb{F}_2^m \to \mathbb{F}_2^n$ be a one-way function. Then, for a given image O of F, we can check whether one of x and $x \oplus \delta$ is a preimage in the naive way by evaluating F on x and $x \oplus \delta$. Now, let us assume that there is a deterministic differential-linear approximation (δ, λ) such that for all $x \in \mathbb{F}_2^m$ the equation $\lambda \cdot (F(x) \oplus F(x \oplus \delta)) = 0$ is fulfilled. Then, for a given image O of F, we can check whether one of x and $x \oplus \delta$ is a preimage in the following way. First, evaluate F on x with $y = F(x)$. If $y = O$, we are done. Otherwise, $\lambda \cdot (y \oplus O) = 0$ is a necessary condition for $x \oplus \delta$ to be a preimage of O. Therefore, we bypass the evaluation of $F(x \oplus \delta)$ when $\lambda \cdot (y \oplus O) \neq 0$. The net effect is that we check 2 messages (x and $x \oplus \delta$) with about 1.5 evaluations of F which speeds up the search. Motivated by this simple idea, we present a general framework for speeding up preimage attacks when multiple highly biased differential-linear approximations are available.

3.1 A General Framework for Speeding up Preimage Attacks

Let $\mathbb{D} = \{\delta_0, \delta_1, \dots, \delta_{s-1}\} \subseteq \mathbb{F}_2^m$ be a set of s *nonzero* differences. For each δ_i $(0 \leq i < s)$, there is a set $\mathbb{M}_i = \{\lambda_{i,0}, \lambda_{i,1}, \dots, \lambda_{i,\ell_i - 1}\}$ of ℓ_i *linear-independent* linear masks, such that each $(\delta_i, \lambda_{i,j})$ forms a DL distinguisher with correlation $\mathfrak{c}_{i,j}$. Algorithm 1 speeds up the search for a preimage from N translations of $\hat{\mathbb{D}}$.

Given an image O of F, Algorithm 1 checks N translations in its N while-loops. In each loop, F is evaluated on a random element x with $y = F(x)$. If $y = O$, we are done. Otherwise, the other s elements in $x \oplus \mathbb{D}$ have to be checked. In a naive approach, s evaluations of F should be performed, including $F(x \oplus \delta_0)$, ..., and $F(x \oplus \delta_{s-1})$. However, according to Line 8 to Line 13 of Algorithm 1, elements in $\{x \oplus \delta_i : \delta_i \in \mathbb{D}, \mathtt{PreTest}(y, O, \delta_i, \mathbb{M}_i) = 1\}$ are rejected without the evaluations of F. To put it another way, only elements in

$$\mathbb{S}_{x,\mathbb{D}} = \{x \oplus \delta_i : \delta_i \in \mathbb{D}, \mathtt{PreTest}(y, O, \delta_i, \mathbb{M}_i) = 0\}$$

are evaluated by F, where $x \oplus \delta_i \in \mathbb{S}_{x,\mathbb{D}}$ is signified by $reject = 0$ in Algorithm 1. The test performed in Line 9 of Algorithm 1 can be regarded as a filtering process. We call $\mathbb{S}_{x,\mathbb{D}}$ the set of *translation survivors*. The saved evaluations of F are the source of the acceleration. Algorithm 1 performs $N(1 + |\mathbb{S}_{x,\mathbb{D}}|)$ evaluations of F, where $|\mathbb{S}_{x,\mathbb{D}}|$ denotes the average size of $\mathbb{S}_{x,\mathbb{D}}$ for a random x. Note that $\mathtt{PreTest}()$ can be implemented with various strategies. For illustration, we first show how

Algorithm 1: Speed up the preimage search with DL distinguishers

Input: $O \in \mathbb{F}_2^n$; The sets of input differences $\mathbb{D} = \{\delta_0, \ldots, \delta_{s-1}\}$ and linear masks $\mathbb{M}_i = \{\lambda_{i,0}, \ldots, \lambda_{i,\ell_i-1}\}$ for $0 \leq i < s$ such that $(\delta_i, \lambda_{i,j})$ is a differential-linear approximation of F with correlation $\mathfrak{c}_{i,j}$

Output: A preimage x such that $F(x) = O$ or $\perp$

$cnt \leftarrow 0$
while $cnt < N$ **do**
  Randomly generate an input $x \in \mathbb{F}_2^m$
  $cnt \leftarrow cnt + 1$
  $y \leftarrow F(x)$
  if $y = O$ **then**
    return x ▷ x is a preimage of O
  for $0 \leq i < s$ **do**
    $reject \leftarrow$ `PreTest`$(y, O, \delta_i, \mathbb{M}_i)$ ▷ Perform some statistical test
    if $reject = 0$ **then**
      $y' \leftarrow F(x \oplus \delta_i)$
      if $y' = O$ **then**
        return $x \oplus \delta_i$
return $\perp$

the complexity and success probability of Algorithm 1 behave under the so-called "strictest" strategy given in Algorithm 2, where we reject an element in a translation whenever one of the differential-linear approximations is not fulfilled.

Algorithm 2: Implement `PreTest`() with the strictest strategy

Input: $y = F(x)$ for some $x \in \mathbb{F}_2^m$, the image O, $\delta_i \in \mathbb{D}$, linear masks $\mathbb{M}_i = \{\lambda_{i,0}, \ldots, \lambda_{i,\ell_i-1}\}$ such that $(\delta_i, \lambda_{i,j})$ is a differential-linear approximation of F with correlation $\mathfrak{c}_{i,j}$

Output: 0 or 1

for $0 \leq j < \ell_i$ **do**
  if $\lambda_{i,j} \cdot (y \oplus O) \neq \zeta_{\mathfrak{c}_{i,j}}$ **then**
    return 1
return 0

Complexity Analysis. When `PreTest`() is instantiated with Algorithm 2, $\mathbb{S}_{x,\mathbb{D}} = \{x \oplus \delta_i : \delta_i \in \mathbb{D}, \lambda_{i,j} \cdot (y \oplus O) = \zeta_{\mathfrak{c}_{i,j}}, 0 \leq j < \ell_i\}$. For each i such that $O \neq F(x \oplus \delta_i)$, the event $\lambda_{i,j} \cdot (y \oplus O) = \zeta_{\mathfrak{c}_{i,j}}$ for all $j \in \{0, \ldots, \ell_i - 1\}$ holds with a probability of $2^{-\ell_i}$. Thus, on average we expect $|\mathbb{S}_{x,\mathbb{D}}| = \sum_{i=0}^{s-1} 2^{-\ell_i}$. Consequently, the complexity of Algorithm 1 is about $N\left(1 + \sum_{i=0}^{s-1} 2^{-\ell_i}\right)$ evaluations of F. Generally, the complexity of the dot products (line 2 of Algorithm 2) is negligible compared with the complexity due to the evaluations of F.

Success Probability. The probability q of hitting a preimage in one while-loop of Algorithm 1 with a random guess $x \in \mathbb{F}_2^m$ can be computed as [3]

$$q \geq \Pr[F(x) = O] + \sum_{i=0}^{s-1} \Pr[F(x \oplus \delta_i) = O \text{ and } x \oplus \delta_i \in \mathbb{S}_{x,\mathbb{D}}]. \quad (1)$$

For $0 \leq i < s$, we have

$$\begin{aligned} & \Pr[F(x \oplus \delta_i) = O \text{ and } x \oplus \delta_i \in \mathbb{S}_{x,\mathbb{D}}] \\ = \; & \Pr[x \oplus \delta_i \in \mathbb{S}_{x,\mathbb{D}} \mid F(x \oplus \delta_i) = O] \; \Pr[F(x \oplus \delta_i) = O] \\ = \; & \Pr[x \oplus \delta_i \in \mathbb{S}_{x,\mathbb{D}} \mid F(x \oplus \delta_i) = O] \left(1 - 2^{-n}\right)^{i+1} \; 2^{-n} \\ = \; & p_i \left(1 - 2^{-n}\right)^{i+1} 2^{-n} > p_i \left(1 - 2^{-n}\right)^{s} 2^{-n}, \end{aligned} \quad (2)$$

where $p_i = \prod_{j=0}^{\ell_i - 1} \left(\frac{1}{2} + \frac{|\mathfrak{c}_{i,j}|}{2}\right)$. Substituting Equation (2) into Equation (1) gives $q > \frac{1}{2^n} + \sum_{i=0}^{s-1} p_i \left(1 - \frac{1}{2^n}\right)^s \frac{1}{2^n}$. Since $s \ll 2^n$ and $\left(1 - \frac{1}{2^n}\right)^s = \left(1 - \frac{1}{2^n}\right)^{2^n \frac{s}{2^n}} \approx e^{-\frac{s}{2^n}} \approx 1$, we have

$$q > \frac{1}{2^n} + \sum_{i=0}^{s-1} \frac{p_i}{2^n} = 2^{\log(s+1)-n} \frac{1}{s+1} \left(1 + \sum_{i=0}^{s-1} p_i\right) = \rho\tau,$$

where $\tau = 2^{\log(s+1)-n}$ and $\rho = \frac{1}{s+1}(1 + \sum_{i=0}^{s-1} p_i)$. Therefore, the success probability that a preimage is detected after N while-loops of Algorithm 1 is lower bounded by $\mathsf{P}_{suc} = 1 - (1 - \rho\tau)^N$. For the sake of comparison with exhaustive search, in this work, we always set $N = (\rho\tau)^{-1}$ to make the success probability to be about $1 - e^{-1} \approx 0.63$, since the success probability to find a preimage of a one way function $F : \mathbb{F}_2^m \to \mathbb{F}_2^n$ by randomly checking 2^n inputs is about $1 - \lim(1 - 1/2^n)^{2^n} \approx 1 - e^{-1} \approx 0.63$. In the above analysis, we assume that the randomly selected translations $x \oplus \hat{\mathbb{D}}$ are disjoint. The following Lemma shows that this assumption is reasonable when m is large.

Lemma 2. *Let $\mathbb{D} \subseteq \mathbb{F}_2^m$ such that $|\mathbb{D}| = s$, and $x_0, \ldots, x_{\alpha-1}$ are randomly generated elements in $\mathbb{F}_2^m$. The probability that the translations $x_i \oplus \hat{\mathbb{D}}$ for $0 \leq i < \alpha$ are not mutually disjoint is upper bounded by $\frac{(s+1)^2\alpha(\alpha-1)}{2^{m+1}}$.*

Proof. See Section A of Supplementary Material. □

For example, in the preimage attack on the 4-round `Ascon-XOF` given in Section 6, we have $m = 320$, $s = 63$, and $\alpha = 2^{122}$. According to Lemma 2, the probability that the randomly generated $\alpha = 2^{122}$ translations are not mutually disjoint is upper bounded by 2^{-65}, which is negligible.

---

[3] For simplicity, we assume that there is one and only one preimage in the search space, i.e., $F(x \oplus \delta_i) = O$ implies $F(x) \neq O$ and $F(x \oplus \delta_j) \neq O$, $j \neq i$.

3.2 Implement PreTest() with More Advanced Statistical Tests

For the sake of completeness and possible further improvement, we introduce some more advanced strategies for implementing PreTest(). However, we strongly encourage the readers first skipping this part since it introduces an additional layer of technical complexity for understanding the core idea. Moreover, since we only use a limited number of DL distinguishers with extremely high correlations in all the concrete cryptanalysis of this work, these more advanced statistical tests do not lead to observable improvements. Therefore, in the applications, we will employ the strictest strategy by default. In addition, with the maximum likelihood strategy and the LLR strategy given in the following, the time complexity of the preimage attack on 4-round Ascon-XOF presented in Section 6 can be marginally improved by a factor of $2^{0.06}$, the details can be found in Section I and Section J of Supplementary Material.

Algorithm 3: Implement PreTest() with the threshold strategy

Input: $y = F(x)$ for some $x \in \mathbb{F}_2^m$, the preimage O, $\delta_i \in \mathbb{D}$, linear masks $\mathbb{M}_i = \{\lambda_{i,0}, \ldots, \lambda_{i,\ell_i-1}\}$ such that $(\delta_i, \lambda_{i,j})$ is a DL approximation of F with correlation $\mathfrak{c}_{i,j}$, and the threshold γ_i

Output: 0 or 1

$num \leftarrow 0$
for $0 \leq j < \ell_i$ **do**
  if $\lambda_{i,j} \cdot (y \oplus O) = \zeta_{\mathfrak{c}_{i,j}}$ **then**
    $num \leftarrow num + 1$
if $num < \gamma_i$ **then**
  return 1
return 0

The Threshold Strategy. In Algorithm 3, an element in a translation is accepted only when there are at least γ_i fulfilled linear approximations. Therefore, the strictest approach given in Algorithm 2 is a special case of the threshold strategy with γ_i set to its maximum (i.e., $\gamma_i = \ell_i$). In this strategy, the complexity of Algorithm 1 is $N(1+\sum_{i=0}^{s-1} q_i)$ evaluations of F, where $q_i = \sum_{z=\gamma_i}^{\ell_i} \binom{\ell_i}{z} 2^{-\ell_i}$. The success probability can be estimated as $1-(1-\rho\tau)^N$, where $\tau = 2^{\log(s+1)-n}$, $\rho = \frac{1}{s+1}(1 + \sum_{i=0}^{s-1} p_i)$, $u = (u_0, \ldots, u_{\ell_i-1}) \in \mathbb{F}_2^{\ell_i}$, and

$$p_i = \sum_{\substack{u \in \mathbb{F}_2^{\ell_i}, \\ wt(u) < \gamma_i}} \prod_{j=0}^{\ell_i-1} \left(\frac{1}{2} + \frac{(-1)^{u_i} \mathfrak{c}_{i,j}}{2} \right).$$

The detailed analysis can be found in Section C.1 of Supplementary Material.

The Maximum Likelihood Strategy. This strategy is implemented in Algorithm 4. We define $L^{(i)} : \mathbb{F}_2^m \mapsto \mathbb{F}_2^{\ell_i}$ to be the function mapping $x \in \mathbb{F}_2^m$ to

$(\lambda_{i,0} \cdot (F(x) \oplus F(x \oplus \delta_i)), \ldots, \lambda_{i,\ell_i-1} \cdot (F(x) \oplus F(x \oplus \delta_i)))$. For $u \in \mathbb{F}_2^{\ell_i}$, let $g_u^{(i)} = \Pr_{x \in \mathbb{F}_2^m}[L^{(i)}(x) = u]$ and $\mathcal{N}_{\gamma_i} = \{u \in \mathbb{F}_2^{\ell_i} : g_u^{(i)} \geq \gamma_i\}$. In this strategy, an element in a translation is accepted if and only if $(\lambda_{i,0} \cdot (y \oplus O), \ldots, \lambda_{i,\ell_i-1} \cdot (y \oplus O)) \in \mathcal{N}_{\gamma_i}$. Therefore, $\mathbb{S}_{x,\mathbb{D}} = \{x \oplus \delta_i : \delta_i \in \mathbb{D}, (\lambda_{i,0} \cdot (F(x) \oplus O), \ldots, \lambda_{i,\ell_i-1} \cdot (F(x) \oplus O)) \in \mathcal{N}_{\gamma_i}\}$, and on average we expect $|\mathbb{S}_{x,\mathbb{D}}| = \sum_{i=0}^{s-1} \frac{|\mathcal{N}_{\gamma_i}|}{2^{\ell_i}}$ for a random x. Consequently, the complexity of Algorithm 1 is about $N\left(1 + \sum_{i=0}^{s-1} \frac{|\mathcal{N}_{\gamma_i}|}{2^{\ell_i}}\right)$ evaluations of F. The success probability can be estimated as $\mathsf{P}_{suc} = 1 - (1 - \rho\tau)^N$, where $\tau = 2^{\log(s+1)-n}$, $\rho = \frac{1}{s+1}(1 + \sum_{i=0}^{s-1} p_i)$, $p_i = \sum_{u \in \mathcal{N}_{\gamma_i}} g_u^{(i)}$. Let $a = (a_0, \ldots, a_{\ell_i-1}) \in \mathbb{F}_2^{\ell_i}$, according to [Lu15, HCN08],

$$g_u^{(i)} = \frac{1}{2^{\ell_i}} \sum_{a \in \mathbb{F}_2^{\ell_i}} (-1)^{a \cdot u} \left(\sum_{x \in \mathbb{F}_2^m} (-1)^{\left(\left(\sum_{j=0}^{\ell_i} a_j \lambda_{i,j}\right) \cdot (F(x) \oplus F(x \oplus \delta_i))\right)} \right).$$

Furthermore, if the differential-linear approximations $(\delta_i, \lambda_{i,j})$, $0 \leq j < \ell_i$ of F with correlation $\mathfrak{c}_{i,j}$ are independent with each other, $g_u^{(i)}$ can be computed as $g_u^{(i)} = \prod_{j=0}^{\ell_i-1} \left(\frac{1}{2} + \frac{(-1)^{u_i} \mathfrak{c}_{i,j}}{2}\right)$, for $u = (u_0, \ldots, u_{\ell_i-1}) \in \mathbb{F}_2^{\ell_i}$. The detailed analysis can be found in Section C.2 of Supplementary Material.

Algorithm 4: A maximum likelihood strategy to implement `PreTest()`

Input: $y = F(x)$ for some $x \in \mathbb{F}_2^m$, the preimage O, $\delta_i \in \mathbb{D}$, linear masks $\mathbb{M}_i = \{\lambda_{i,0}, \ldots, \lambda_{i,\ell_i-1}\}$ such that $(\delta_i, \lambda_{i,j})$ is a differential-linear approximation of F with correlation $\mathfrak{c}_{i,j}$, and the set $\mathcal{N}_{\gamma_i}$.
Output: 0 or 1

```
v ← (λ_{i,0} · (y ⊕ O), ..., λ_{i,ℓ_i−1} · (y ⊕ O))
if v ∈ N_{γ_i} then
  | return 0
else
  ⌊ return 1
```

The LLR Strategy. See Section C.3 of Supplementary Material.

4 The Framework for Speeding up Key-Recovery Attacks

Let $F : \mathbb{F}_2^m \times \mathbb{F}_2^n \to \mathbb{F}_2^n$ be a keyed function with $F(K, P) = C$ and K being the secret key. In typical key-recovery attacks, the adversary is free to make her own choices of P. Suppose that we have a deterministic related-key DL distinguisher such that $\lambda \cdot (F(K, P) \oplus F(K \oplus \delta, P \oplus \delta')) = 0$. In addition, $C = F(K, P)$ and $C' = F(K, P \oplus \delta')$. Then, we can check whether one of k and $k \oplus \delta$ is a candidate key by computing $c = F(k, P)$. If $c = C$, then k is a candidate for the correct key. Also, $\lambda \cdot (c \oplus C') = 0$ is necessary for $k \oplus \delta$ being a candidate for K with $k \oplus \delta = K$, since in this case $\lambda \cdot (c \oplus C') = \lambda \cdot (F(k, P) \oplus F(k \oplus \delta, P \oplus \delta')) = 0$ according to the deterministic distinguisher. Therefore, we bypass the evaluation of $F(k \oplus \delta, P \oplus \delta')$ when $\lambda \cdot (c \oplus C') \neq 0$. The net result is that we check 2 keys (k and $k \oplus \delta$) with about 1.5 evaluations of F. We emphasize that in the whole process, we do not query the encryption oracle with $K \oplus \delta$. Therefore, *the attack exploits related-key differential-linear distinguishers in the single-key model.*

Given (P, C), the function $F_P(\cdot) = F(\cdot, P)$ can be regarded as a one-way function parameterized by P. Therefore, the preimage K for C can be recovered with similar methods given in Section 3. For simplicity, we just give one full example in Algorithm 5 with the strictest strategy for the statistical test described in Algorithm 6. Let $\mathbb{D} = \left\{(\delta_0, \delta_0'), (\delta_1, \delta_1'), \dots, (\delta_{s-1}, \delta_{s-1}')\right\} \subseteq \mathbb{F}_2^{m+n}$ be a set of s differences where $\delta_i \neq 0$ $(0 \leq i < s)$, and for each difference (δ_i, δ_i'), there is a set $\mathbb{M}_i = \{\lambda_{i,0}, \lambda_{i,1}, \dots, \lambda_{i,\ell_i-1}\}$ of ℓ_i linearly-independent linear masks. Each $((\delta_i, \delta_i'), \lambda_{i,j})$ forms a DL distinguisher with correlation $\mathfrak{c}_{i,j}$. In contrast to preimage attacks (Algorithm 1) where the search terminates whenever a preimage is identified, a key-recovery attack has to find the actual key used. To this end, Algorithm 5 requires that $\hat{\mathbb{D}}_K = \mathbb{D}_K \cup \{0\} = \{0, \delta_0, \delta_1, \dots, \delta_{s-1}\}$ to be a linear space. In this way, the full key space $\mathbb{F}_2^m$ is covered by the $2^{m-\dim(\hat{\mathbb{D}}_K)} = 2^{m-\log(s+1)}$ translations of $\hat{\mathbb{D}}_K$ (Lemma 1). Algorithm 5 checks these translations in its $2^{m-\log(s+1)}$ for-loops. In each loop, F is evaluated on a random key k with $c = F(k, P)$. If $c = C$, k is a key candidate, and we confirm it with additional plaintext-ciphertext pairs. Otherwise, the other s elements in $k \oplus \hat{\mathbb{D}}_K$ have to be checked. Similarly to Algorithm 1, only elements in the set of translation survivors $\mathbb{S}_{k,\mathbb{D}_K} = \{k \oplus \delta_i : \delta_i \in \mathbb{D}_K, \texttt{KeyTest}(c, C_i, (\delta_i, \delta_i'), \mathbb{M}_i) = 0\}$ are evaluated by F, where $k \oplus \delta_i \in \mathbb{S}_{k,\mathbb{D}_K}$ is signified by $reject = 0$ in Algorithm 6. Again, the saved evaluations of F are the source of the acceleration.

Complexity Analysis. When $\texttt{KeyTest}()$ is instantiated with Algorithm 6, $\mathbb{S}_{k,\mathbb{D}_K} = \{k \oplus \delta_i : \delta_i \in \mathbb{D}, \lambda_{i,j} \cdot (c \oplus C_i) = \zeta_{\mathfrak{c}_{i,j}}, 0 \leq j < \ell_i\}$. In each for-loop of Algorithm 5 (line 4), F is evaluated once on a randomly selected $k \in \hat{\mathbb{D}}_K^{\dashv}$ to encrypt a plaintext P, and then F is evaluated $|\mathbb{S}_{k,\mathbb{D}_K}|$ times. For each i such that $C_i \neq F(k \oplus \delta_i, P \oplus \delta_i')$, the event $\lambda_{i,j} \cdot (c \oplus C_i) = \zeta_{\mathfrak{c}_{i,j}}$ for all $j \in \{0, \cdots, \ell_i - 1\}$ holds with a probability of $2^{-\ell_i}$. Thus, on average we expect $|\mathbb{S}_{k,\mathbb{D}_K}| = \sum_{i=0}^{s-1} 2^{-\ell_i}$. Consequently, the complexity of Algorithm 5 is about $2^{\dim(\hat{\mathbb{D}}_K^{\dashv})}(1 + \sum_{i=0}^{s-1} 2^{-\ell_i})$ evaluations of F, where $\dim(\hat{\mathbb{D}}_K^{\dashv}) = m - \log(s+1)$ according to Lemma 1.

Success Probability. Since the translations $k \oplus \hat{\mathbb{D}}_K$ of $\hat{\mathbb{D}}_K$ with $k \in \hat{\mathbb{D}}_K^{\dashv}$ form a partition of $\mathbb{F}_2^m$, the correct key K must be in one of the translations for some k, where K is randomly chosen from $\mathbb{F}_2^m$. The probability q of hitting the correct key by Algorithm 5 can be estimated as

$$q = \frac{1}{s+1} + \sum_{i=0}^{s-1} \frac{p_i}{s+1} = \frac{1}{s+1}\left(1 + \sum_{i=0}^{s-1} p_i\right), \tag{3}$$

Algorithm 5: Speed up the key-recovery with DL distinguishers

Input: $\mathbb{D} = \{(\delta_0, \delta_0'), \cdots, (\delta_{s-1}, \delta_{s-1}')\} \subseteq \mathbb{F}_2^{m+n}$, and $\mathbb{M}_i = \{\lambda_{i,0}, \ldots, \lambda_{i,\ell_i-1}\}$ for $0 \le i < s$ such that $((\delta_i, \delta_i'), \lambda_{i,j})$ is a *related-key* DL appoximation of F with correlation $\mathfrak{c}_{i,j}$, and $\hat{\mathbb{D}}_K = \{0\} \cup \{\delta_0, \cdots, \delta_{s-1}\}$ is a linear subspace of $\mathbb{F}_2^m$.

Output: The master key K

Randomly choose a plaintext P, derive $C = F(K, P)$
for $0 \le i < s$ **do**
  $C_i = F(K, P \oplus \delta_i')$
for $k \in \hat{\mathbb{D}}_K^{\dashv}$ **do**
  $c \leftarrow F(k, P)$
  if $c = C$ **then**
    if $F(k, P \oplus \delta_i') = C_i, 0 \le i < s$ **then**
      return k ▷ `a few of` $(P \oplus \delta_i', C_i)$ `suffice`
  for $0 \le i < s$ **do**
    reject $\leftarrow$ `KeyTest`$(c, C_i, (\delta_i, \delta_i'), \mathbb{M}_i)$
    if *reject* $= 0$ **then**
      if $F(k \oplus \delta_i, P \oplus \delta_i') = C_i, 1 \le i < s$ **then**
        return $k \oplus \delta_j$ ▷ `a few of` $(P \oplus \delta_i', C_i)$ `suffice`

Algorithm 6: A strictest approach to implement `KeyTest()`

Input: $c = F(k, P)$, $C_i = F(K, P \oplus \delta_i')$, $(\delta_i, \delta_i') \in \mathbb{F}_2^m \times \mathbb{F}_2^n$, and $\mathbb{M}_i = \{\lambda_{i,0}, \ldots, \lambda_{i,\ell_i-1}\}$ such that $((\delta_i, \delta_i'), \lambda_{i,j})$ is a *related-key* DL appoximation of F with correlation $\mathfrak{c}_{i,j}$

Output: 0 or 1

for $0 \le j < \ell_i$ **do**
  if $\lambda_{i,j} \cdot (c \oplus C_i) \ne \zeta_{\mathfrak{c}_{i,j}}$ **then**
    return 1
return 0

where $p_i = \prod_{j=0}^{\ell_i-1}\left(\frac{1}{2} + \frac{|\mathfrak{c}_{i,j}|}{2}\right)$. The detailed analysis can be found in Section D of Supplementary Material. If we only use *deterministic* DL distinguishers as we do in all of our concrete cryptanalysis in this paper, the success probability of Algorithm 5 is about 1.

Remark 2. In Algorithm 5, we require $\hat{\mathbb{D}}_K$ to be a linear space. Although in all concrete applications in the following sections this condition is fulfilled, theoretically, this condition is not necessary. If the $\hat{\mathbb{D}}_K = \{0, \delta_0, \delta_1, \ldots, \delta_{s-1}\}$ is not a linear subspace, the key search can be accelerated with Algorithm 9 given in Section E of the Supplementary Material.

Next, we show how to use a large hash table to speed-up the key search when there are a huge number of differential-linear approximations by a contrived example. This technique is applied in the analysis of `Schwaemm` in Section 8, and Section N in Supplementary Material. Let $F : \mathbb{F}_2^{256} \times \mathbb{F}_2^{256} \to \mathbb{F}_2^{256}$ be a keyed function mapping (K, P) to $C = F(K, P)$. Let $\mathbb{D} = \{(\delta'_0, \delta_0), \cdots, (\delta'_{2^{128}-2}, \delta_{2^{128}-2})\}$ be a set of differences, such that $\hat{\mathbb{D}}_K = \{0, \delta_0, \cdots, \delta_{2^{128}-2}\}$ forms a linear space with dimension 128. Let $\mathbb{M}_i = \{\lambda_0, \cdots, \lambda_{127}\}$ be a set of linear masks for $0 \leq i < 2^{128} - 1$, such that $\lambda_j \cdot (F(k \oplus \delta_i, P \oplus \delta'_i) \oplus F(k, P)) = 0$ for all $j \in \{0, \cdots, 127\}$ and $i \in \{0, \cdots, 2^{128} - 2\}$ deterministically. In addition, let $L = (\lambda_0, \cdots, \lambda_{127})^T$, which can be regarded as a 128×256 binary matrix (i.e., a linear transformation). The procedure of the attack is given in Algorithm 7. Firstly, from Line 3 to Line 5 of Algorithm 7, we need to evaluate 2^{128} times F and 2^{128} times L. In each for-loop at Line 6, on average, we need to perform F (Line 7) one time, L (Line 10) one time, F (Line 12) one time on average, and one hash table lookup. Since the for loop will repeat $2^{256-128}$ times, we need to perform $2^{128} + 2^{128}$ times F, 2^{128} times L, and 2^{128} hash table lookups. Therefore, the time complexity is 3×2^{128} times F, 2×2^{128} times L, and 2^{128} times hash table lookups.

Algorithm 7: Speed up key-recovery attacks with hash tables

Input: $\mathbb{D}$ and $\mathbb{M}_i$ for $0 \leq i < 2^{128} - 1$
Output: The master key K

1 Randomly choose a plaintext P
2 $C \leftarrow F(K, P)$ `// Query the oracle`
3 **for** $0 \leq i < 2^{128} - 1$ **do**
4     $C_{\delta_i} \leftarrow F(K, P \oplus \delta'_i)$ `// Query the oracle`
5   Insert δ_i into a hash table at address $L(C_{\delta_i})$
6 **for** $k \in \hat{\mathbb{D}}_K^{\dashv}$ **do**
7     $c \leftarrow F(k, P)$
8   **if** $c = C$ **then**
9   **return** k
10   `Addr` $\leftarrow L(c)$
11   **for** δ *at address* `Addr` *of the hash table* **do**
12        $C' \leftarrow F(k \oplus \delta, P)$
13   **if** $C' = C$ **then**
14   **return** $k \oplus \delta$
15 **return** $\perp$

5 Application I: Preimage Attacks on XOEsch

XOEsch512 and XOEsch384 are two XOFs of the NIST LWC finalist Sparkle family built with the sponge structure, whose parameters and security bounds are listed in Table 3. The structure of XOEsch384 is given as an example in Figure 1. According to the specification of XOEsch, only when necessary, the message is padded. Thus, we always assume that there are no padding bits in the message in our attacks. Also, different from most sponge-based hash functions, XOEsch applies an $\mathcal{M}_w$ operation to its message blocks before absorbing them.

Table 3: Parameters used by XOEsch256 and XOEsch384 with the digest length being $t > 0$. Our attacks are applied to the cases with $t = 128$ and $t = 192$.

Instance	Size			Security Claim	
	Permutation	Rate	Capacity	Collision	(2nd) Preimage
XOEsch256	384	128	256	$\min\{128, t/2\}$	$\min\{128, t\}$
XOEsch384	512	128	384	$\min\{192, t/2\}$	$\min\{192, t\}$

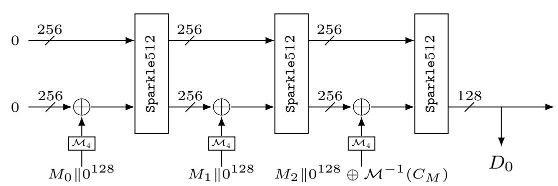


Fig. 1: The structure of XOEsch384. In this example, the input is a 3-block message (M_0, M_1, M_2), the output is a 1-block digest D_0. C_M is a constant to differentiate different instances of Esch and XOEsch.

Definition 1. *Let $w > 1$ be an integer. $\mathcal{M}_w$ permutes $(\mathbb{F}_2^{32} \times \mathbb{F}_2^{32})^w$ such that*

$$\mathcal{M}_w\left((x_0, y_0), \ldots, (x_{w-1}, y_{w-1})\right) = \left((u_0, v_0), \ldots, (u_{w-1}, v_{w-1})\right),$$

where the branches (u_i, v_i) are defined as

$$t_y \leftarrow \bigoplus_{i=0}^{w-1} y_i, \quad t_x \leftarrow \bigoplus_{i=0}^{w-1} x_i, \quad \begin{cases} u_i \leftarrow x_i \oplus \ell(t_y) \\ v_i \leftarrow y_i \oplus \ell(t_x) \end{cases} \forall i \in \{0, \ldots, w-1\},$$

where ℓ is a linear operation.

Lemma 3 (Fixed Point). *Let $z_i \in \mathbb{F}_2^{64}$ for $0 \le i < w$. If $\bigoplus_{i=0}^{w-1} z_i = 0$, then $\mathcal{M}_w(z_0, z_1, \ldots, z_{w-1}) = (z_0, z_1, \ldots, z_{w-1})$.*

Proof. According to Definition 1, $\bigoplus_{i=0}^{w-1} z_i = 0$ implies $t_x = t_y = 0$. ◻

Lemma 4 (Mask Invariance). *Let $\lambda \in \mathbb{F}_2^{64} \backslash \{0\}$ and $z_i \in \mathbb{F}_2^{64}$ for $0 \le i < w$. Define $\gamma = (\gamma_0, \gamma_1, \dots, \gamma_{w-1}) \in \mathbb{F}_2^{64\times w}$ and $\gamma_i \in \{0, \lambda\}$ $(0 \le i < w)$. If there are even-number out of the w components of γ being $\lambda \neq 0$, we have*

$$\gamma \cdot (z_0, z_1, \dots, z_{w-1}) = \gamma \cdot (z'_0, z'_1, \dots, z'_{w-1}),$$

where $(z'_0, z'_1, \dots, z'_{w-1}) = \mathcal{M}_w(z_0, z_1, \dots, z_{w-1})$.

Proof. See Section B of Supplementary Material. ◻

Lemma 5 (Valid Input). *Let $(u_i, v_i) \in \mathbb{F}_2^{32} \times \mathbb{F}_2^{32}$ for $0 \le i < w$ such that*

$$(u_i, v_i) = (\ell(v_0 \oplus v_1), \ell(u_0 \oplus u_1)) \text{ for } 2 \le i < w.$$

Then, there is a unique $((x_0, y_0), (x_1, y_1), (0, 0), \dots, (0, 0)) \in (\mathbb{F}_2^{32} \times \mathbb{F}_2^{32})^w$ satisfying $\mathcal{M}_w((x_0, y_0), (x_1, y_1), (0, 0), \dots, (0, 0)) = ((u_0, v_0), \dots, (u_{w-1}, v_{w-1}))$, where

$$\begin{cases} (x_0, y_0) = (u_0 \oplus \ell(v_0 \oplus v_1), v_0 \oplus \ell(u_0 \oplus u_1)) \\ (x_1, y_1) = (u_1 \oplus \ell(v_0 \oplus v_1), v_1 \oplus \ell(u_0 \oplus u_1)) \end{cases}. \quad (4)$$

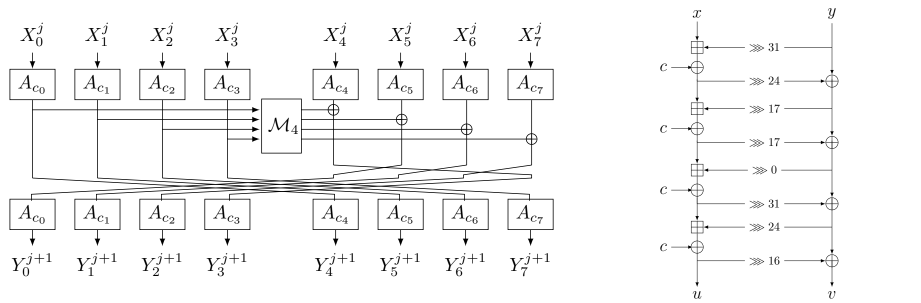
Proof. See Section B of Supplementary Material. ◻

`Sparkle` is a family of ARX-based permutations used by the hash functions `XOEsch` as well as its XOFs `XOEsch` and the AEAD `Schwaemm` [BBdS+21]. Figure 2a illustrates the structure of the 1.5-step `Sparkle`512 permutation, where $A_{c_i} : \mathbb{F}_2^{32} \times \mathbb{F}_2^{32} \to \mathbb{F}_2^{32} \times \mathbb{F}_2^{32}$ is an ARX box parameterized with a constant c_i named as `Alzette` (see Figure 2b). For convenience, the input and output of the j-th step of the `Sparkle` permutation are denoted by $X^j = (X_0^j, \dots, X_{z-1}^j)$ and $Y^j = (Y_0^j, \dots, Y_{z-1}^j)$, where $z = 4, 6, 8$ for `Sparkle`256, `Sparkle`384 and `Sparkle`512, respectively (see Figure 2a as an example).

We apply Algorithm 1 and give the preimage attacks on 1.5- and 2.5-step `XOEsch`384 in Sections 5.2 and 5.3, respectively. In Section F of Supplementary Material, we give the preimage attacks on 1.5- and 2.5-step `XOEsch`256. In Section G, we provide preimage attacks on variants of `XOEsch`384 and `XOEsch`256 where the two `Alzette` ARX boxes in the first round are parameterized with the same constants. The results justify the choice of the designers to use different constants to parameterize different `Alzette` ARX boxes.

5.1 DL Distinguishers for `Alzette`

We identify 15 groups of DL distinguishers with the method given in [NSLL22], which are listed in Table 4. The absolute correlations of these distinguishers are extremely high and have been verified experimentally. Let the set of input differences of A_c be a linear space $\hat{\mathbb{D}}_{\mathsf{Alzette}}$ spanned by $b_0 = (\mathtt{0x80000000}, \mathtt{0x0})$, $b_1 = (\mathtt{0x40000000}, \mathtt{0x0})$, $b_2 = (\mathtt{0x20000000}, \mathtt{0x0})$, and $b_3 = (\mathtt{0x0}, \mathtt{0x40000000})$.



(a) The structure of 1.5-step of `Sparkle`512 permutation. In this instance, there are 8 64-bit branches.

(b) `Alzette` parameterized by c.

Fig. 2: Illustration of `Sparkle` and `Alzette`.

Thus, $\hat{\mathbb{D}}_{\mathsf{Alzette}} = \langle b_0, b_1, b_2, b_3 \rangle$, and any difference $\delta \in \hat{\mathbb{D}}_{\texttt{Alzette}}$ can be written as $\delta = \sum_{i=0}^{3} a_i b_i$ where $a_i \in \mathbb{F}_2$, denoted by $(a_0a_1a_2a_3)_\delta$. For example, since $(\texttt{0xa0000000}, \texttt{0x40000000}) = b_0 \oplus b_1 \oplus b_3$, $(\texttt{0xa0000000}, \texttt{0x40000000})$ is accordingly denoted by $(\texttt{1101})_\delta$. Each linear mask of A_c we used has only two active bits: one is in the left branch, and the other is in the right. So we use the two indices of the active bits in the left and right branches to denote the masks. For example, $(\texttt{0}, \texttt{31})_\lambda$ represents the mask $(\texttt{0x80000000}, \texttt{0x1})$.

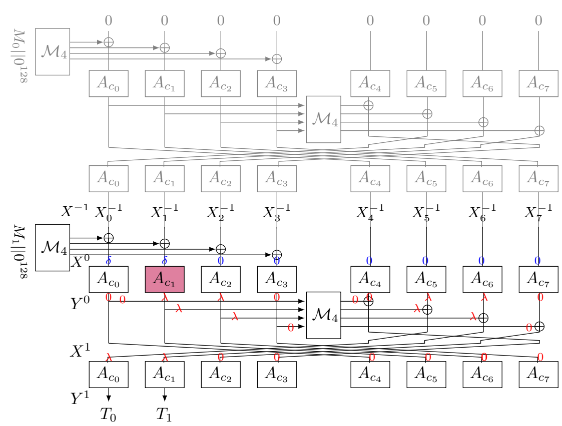


Fig. 3: Preimage attack on the 1.5-step `XOEsch`384 with a 128-bit digest. We use two message blocks (M_0, M_1) in this attack.

Table 4: The DL distinguishers of A_c with *absolute* correlations. These input differences form $\mathbb{D}_{\texttt{Alzette}}$ and $\hat{\mathbb{D}}_{\texttt{Alzette}}$ is a linear space. All or the first five linear masks in the table head form $\mathbb{M}_i$ for each $\delta_i \in \mathbb{D}_{\texttt{Alzette}}$.

Diff.\Mask	$(17,1)_\lambda$	$(18,2)_\lambda$	$(19,3)_\lambda$	$(5,21)_\lambda$	$(4,20)_\lambda$	$(14,30)_\lambda$	$(28,12)_\lambda$
$(0010)_\delta$	1	1	1	≥ 0.96	≥ 0.94	≥ 0.96	≥ 0.90
$(0100)_\delta$	1	1	1	≥ 0.96	≥ 0.94	≥ 0.94	≥ 0.86
$(1000)_\delta$	1	1	1	≥ 0.96	≥ 0.92	≥ 0.92	≥ 0.82
$(0110)_\delta$	1	1	1	≥ 0.96	≥ 0.94	≥ 0.94	≥ 0.88
$(1010)_\delta$	1	1	1	≥ 0.96	≥ 0.94	≥ 0.94	≥ 0.84
$(1100)_\delta$	1	1	1	≥ 0.96	≥ 0.94	≥ 0.94	≥ 0.86
$(1110)_\delta$	1	1	1	≥ 0.96	≥ 0.94	≥ 0.94	≥ 0.88
$(0001)_\delta$	≥ 0.92	≥ 0.92	≥ 0.94	≥ 0.916	≥ 0.84		
$(0011)_\delta$	≥ 0.92	≥ 0.92	≥ 0.94	≥ 0.92	≥ 0.84		
$(0101)_\delta$	≥ 0.92	≥ 0.92	≥ 0.94	≥ 0.92	≥ 0.84		
$(0111)_\delta$	≥ 0.92	≥ 0.92	≥ 0.94	≥ 0.92	≥ 0.84		
$(1011)_\delta$	≥ 0.92	≥ 0.92	≥ 0.94	≥ 0.92	≥ 0.84		
$(1101)_\delta$	≥ 0.92	≥ 0.92	≥ 0.94	≥ 0.92	≥ 0.84		
$(1111)_\delta$	≥ 0.92	≥ 0.92	≥ 0.94	≥ 0.92	≥ 0.84		
$(1001)_\delta$	≥ 0.92	≥ 0.92	≥ 0.94	≥ 0.92	≥ 0.86		

5.2 Preimage Attack on the 1.5-Step `XOEsch384`

Our preimage attack works for the 1.5-step `XOEsch384` with a digest length between 128 and 192 bits, and to ensure the disjointness of the generated translations, it requires 2 message blocks (M_0, M_1). Here we take the instance with a 128-bit digest illustrated in Figure 3 as an example. The 128-bit digest $(T_0, T_1) \in \mathbb{F}_2^{64\times 2}$ can be inverted through `Alzette` ARX boxes A_{c_0} and A_{c_1}. Thus, if the linear masks employed for $(X_2^1, \dots, X_7^1)$ in the attack are inactive, we can safely skip the `Alzette` ARX boxes in the last step. In addition, for any given M_0, X^{-1} can be derived. Consequently, we only need to focus on the function $F_{LSM} : \mathbb{F}_2^{128} \to \mathbb{F}_2^{128}$ mapping M_1 to (X_0^1, X_1^1).

The DL approximations for F_{LSM} are derived from DL distinguishers of `Alzette`. Given any DL approximation (δ, λ) of A_{c_1} with correlation $\mathfrak{c}$ listed in Table 4, we set the linear mask of X^1 to be $\Lambda(X^1) = (\lambda, \lambda, 0, 0, 0, 0, 0, 0)$. According to Lemma 4, the linear mask $\Lambda(Y^0)$ of Y^0 is $(0, \lambda, \lambda, 0, 0, \lambda, \lambda, 0)$. Let the difference of M_1 be $\Delta(M_1) = (\delta, \delta)$. According to Lemma 3, the difference of X^0 is $\Delta(X^0) = (\delta, \delta, 0, 0, 0, 0, 0, 0)$. As highlighted in Figure 3, only A_{c_1} has nonzero input difference and nonzero output linear mask at the same time. Therefore, the correlation of the above DL approximation for F_{LSM} is $\mathfrak{c}$.

The attack applies Algorithm 1 to F_{LSM} and proceeds as follows in the t-th while-loop of Algorithm 1. Set M_0 to be the 128-bit encoding of the integer t, and generate one random message block $M_1 \in \mathbb{F}_2^{128}$. Compute the value $\boldsymbol{x} = (x_0, x_1)$ for (X_0^1, X_1^1) from M_0 and M_1. If $\boldsymbol{x} = (x_0, x_1) = (A_{c_0}^{-1}(T_0), A_{c_1}^{-1}(T_1)) =$

(X_0^1, X_1^1), we are done with (M_0, M_1) being the preimage of (T_0, T_1). Otherwise, for each $\delta_i \in \mathbb{D}_{\texttt{Alzette}}$, we test whether $\lambda_{i,j} \cdot (\boldsymbol{x} \oplus (X_0^1, X_1^1)) = \zeta_{\mathfrak{c}_{i,j}}$ for all $\lambda_{i,j} \in \mathbb{M}_i$ ($\mathbb{D}_{\texttt{Alzette}}$ and $\mathbb{M}_i$ are given in Table 4). If δ_i passes the test, we compute the value $\boldsymbol{x}' = (x_0', x_1')$ for (X_0^1, X_1^1) from the message $(M_0, M_1 \oplus (\delta_i, \delta_i))$. If $\boldsymbol{x}' = (x_0', x_1') = (A_{c_0}^{-1}(T_0), A_{c_1}^{-1}(T_1))$, $(M_0, M_1 \oplus (\delta_i, \delta_i))$ is a preimage for (T_0, T_1). Note that with our approach for selecting (M_0, M_1), the translations checked in the first N while-loops with $N < 2^{128}$ are guaranteed to be disjoint since the first 128 bits of two messages in the translations checked in different while-loops encode different integers.

Complexity and Success Probability. According to Table 4, the size of the set $\mathbb{D}$ of input differences is $s = |\mathbb{D}| = |\mathbb{D}_{\texttt{Alzette}}| = 15$, $\rho \approx 2^{-0.26}$ and $\tau = 2^{\log(s+1)-n} = 2^{-124}$. The expectation of $|\mathbb{S}_{x,\mathbb{D}}| = \sum_{i=0}^{s-1} 2^{-\ell_i}$ is about $2^{-1.71}$. Therefore, we set $N = (\rho\tau)^{-1} = 2^{124.26}$ to make the success probability to be 0.63. The time complexity of the attack is about $N(1 + 2^{-1.71}) = 2^{124.26} \times (1 + 2^{-1.71}) \approx 2^{124.64}$ evaluations of F_{LSM}.

In our attack, the selection of the $N = 2^{124.26}$ translations can be optimized by randomly choosing, e.g., $2^{100.26}$ M_0 and under each chosen M_0 we choose 2^{24} M_1 randomly. With this technique, the computation of M_0 is negligible compared to other parts. Moreover, considering that the nonlinear operations in `XOEsch` is much more costly than the linear layer, we approximately regard the cost of F_{LSM} as that of one step of `Sparkle`512. The 1.5-step `XOEsch`384 instance with a 128-bit digest requires about one 1.5-step `Sparkle`512 (2 nonlinear layers). Consequently, the complexity of the attack is approximately $2^{123.64}$ 1.5-step `XOEsch`384 evaluations.

Complexity for the `XOEsch384` with a 192-bit Digest. The digest of this instance consists of 2 blocks (one is 128-bit and the other is 64-bit) generated by two iterations of the 1.5-step `Sparkle` permutation. We perform a similar attack to match the first block, and only when the first match is successful we continue to match the second block. Since the probability of the first matching is very low, the cost for the second matching is negligible. The complexity is about $2^{186.64}$ 1.5-step `XOEsch`384 calculations whose success probability is at least 0.63.

5.3 Preimage Attack on the 2.5-Step `XOEsch384`

Our second application is to the 2.5-step `XOEsch`384. Akin to the preimage attack on the 1.5-step `XOEsch`384, we take the 128-bit-digest instance of `XOEsch`384 as an example. To ensure the disjointness of the generated translations, this attack requires 2^{127} translations of a 1-dimensional linear space, so we use 2 message blocks denoted by (M_0, M_1) (see Figure 4).

The 128-bit digest $(T_0, T_1) \in \mathbb{F}_2^{64\times 2}$ can be inverted through `Alzette` ARX boxes A_{c_0} and A_{c_1}. Thus, if the linear masks employed for $(X_2^2, \ldots, X_7^2)$ in the attack are inactive, we can safely skip the `Alzette` ARX boxes in the last step. In addition, when we choose an M_0, X^{-1} will be obtained. Consequently, in our preimage attack on the 2.5-step `XOEsch`384, we only need to focus on the second message block, i.e., M_1. Different from the 1.5-step attack, the function

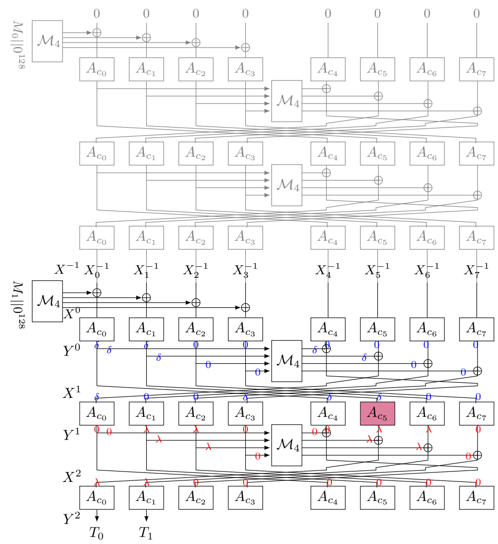


Fig. 4: Illustration of the preimage attacks on the 2.5-step XOEsch384 with a 128-bit digest. This attack also uses a 2-block message (M_0, M_1).

that we apply Algorithm 1 to is $F_{LSL} : \mathbb{F}_2^{256} \to \mathbb{F}_2^{128}$ mapping $(Y_0^0, Y_1^0, Y_2^0, Y_3^0)$ to (X_0^2, X_1^2), rather than function sending M_1 to (X_0^2, X_1^2), because the 2.5-step Sparkle512 is more complicated and more difficult to allow DL distinguishers.

Next, we introduce the DL distinguishers for F_{LSL}. Given any DL approximation (δ, λ) of A_{c_5} with correlation $\mathfrak{c}$, we set the linear mask of X^2 to be $\Lambda(X^2) = (\lambda, \lambda, 0, 0, 0, 0, 0, 0)$. According to Lemma 4, the linear mask $\Lambda(Y^1)$ of Y^1 is $(0, \lambda, \lambda, 0, 0, \lambda, \lambda, 0)$. For the difference of Y^0, we set it to be $\Delta(Y^0) = (\delta, \delta, 0, 0, 0, 0, 0, 0)$. The difference of X^1 will be $\Delta(X^1) = (\delta, 0, 0, \delta, \delta, \delta, 0, 0)$, according to Lemma 3. Now, as highlighted in Figure 4, only the input difference and output linear mask of A_{c_5} in the second step are both nonzero. Therefore, the correlation of the above DL approximation for F_{LSL} is $\mathfrak{c}$.

When applying Algorithm 1 to F_{LSL}, under each M_0 that we have chosen, we need to guess and check a value for $(Y_0^0, Y_1^0, Y_2^0, Y_3^0)$, say $\boldsymbol{y} = (y_0, y_1, y_2, y_3)$, and quickly check $\boldsymbol{y}' = (y_0 \oplus \delta, y_1 \oplus \delta, y_2, y_3)$ with the DL distinguishers. In this process, both $\boldsymbol{y}$ and $\boldsymbol{y}'$ are possible to be a preimage of (T_0, T_1). However, due to the existence of $\mathcal{M}_4$ in the absorption phase and more critically, the second 128-bit input of this $\mathcal{M}_4$ should be 0 (see Figure 1), there is a risk that the recovered $\boldsymbol{y}$ or $\boldsymbol{y}'$ does not correspond to any valid M_1.

To address this risk, we do pre-computations to search for some 3-tuples $(\gamma_0, \gamma_1, \delta)$ satisfying

$$A_{c_0}^{-1}(\gamma_0) \oplus A_{c_0}^{-1}(\gamma_0 \oplus \delta) = A_{c_1}^{-1}(\gamma_1) \oplus A_{c_1}^{-1}(\gamma_1 \oplus \delta). \tag{5}$$

When X^{-1} is known, based on any pre-computed $(\gamma_0, \gamma_1, \delta)$ we can choose $\boldsymbol{y}$ and $\boldsymbol{y}'$ such that both $\boldsymbol{y}$ and $\boldsymbol{y}'$ can lead to a valid M_1 in the following way,

$$\begin{cases} \boldsymbol{y} = (y_0, y_1, y_2, y_3) = (\gamma_0, \gamma_1, \gamma_2, \gamma_3) \\ \boldsymbol{y}' = (y_0 \oplus \delta, y_1 \oplus \delta, y_2, y_3) = (\gamma_0 \oplus \delta, \gamma_1 \oplus \delta, \gamma_2, \gamma_3) \end{cases} \tag{6}$$

where

$$\begin{cases} (u_j, v_j) = A_{c_j}^{-1}(\gamma_j) \oplus X_j^{-1}, j \in \{0, 1\} \\ \gamma_i = A_{c_i}\left((\ell(v_0, v_1), \ell(u_0, u_1)) \oplus X_i^{-1}\right), i \in \{2, 3\} \end{cases}.$$

It can be checked $\boldsymbol{y} = (y_0, y_1, y_2, y_3)$ and $\boldsymbol{y}' = (y_0 \oplus \delta, y_1 \oplus \delta, y_2, y_3)$ respectively guarantee that $(A_{c_0}^{-1}(y_0), A_{c_1}^{-1}(y_1), A_{c_2}^{-1}(y_2), A_{c_3}^{-1}(y_3)) \oplus (X_0^{-1}, X_1^{-1}, X_2^{-1}, X_3^{-1})$ and $(A_{c_0}^{-1}(y_0 \oplus \delta), A_{c_1}^{-1}(y_1 \oplus \delta), A_{c_2}^{-1}(y_2), A_{c_3}^{-1}(y_3)) \oplus (X_0^{-1}, X_1^{-1}, X_2^{-1}, X_3^{-1})$ satisfy Lemma 5 (it this case, the w in Lemma 5 should be instanced as 4). Hence, no matter whether Algorithm 1 returns $\boldsymbol{y}$ from Line 7 or $\boldsymbol{y}'$ from Line 13, we are sure that M_1 exists.

In terms of the pre-computation, we simply try different $\gamma_0, \gamma_1 \in \mathbb{F}_2^{64}$ for one given δ to see if Equation (5) holds. When it holds, $(\gamma_0, \gamma_1, \delta)$ is one such 3-tuple we need. Since Equation (5) holds with a probability of about 2^{-64} (two 64-bit values are equal), trying 2^{74} different (γ_0, γ_1) for one δ, we can expect to collect 2^{10} $(\gamma_0, \gamma_1, \delta)$. For the sake of convenience, we put all these collected 3-tuples into a table $\mathbb{S}_\delta$. Equation (5) implies that it is impossible to use more than one different δ's, because for t different δ's, we need to find (γ_0, γ_1) to satisfy t Equation (5) with different δ's, which has a probability of 2^{-64t}. (γ_0, γ_1) has at most 2^{128} possibilities, so t has to be 1. For this attack, we choose $\delta = (\mathtt{1001})_\delta$ that has a group of DL distinguishers as

$$\delta = (\mathtt{1001})_\delta, \mathbb{M} = \begin{Bmatrix} (\mathtt{25}, \mathtt{9})_\lambda, (\mathtt{26}, \mathtt{10})_\lambda, (\mathtt{27}, \mathtt{11})_\lambda, (\mathtt{28}, \mathtt{12})_\lambda, \\ (\mathtt{29}, \mathtt{13})_\lambda, (\mathtt{30}, \mathtt{14})_\lambda, (\mathtt{11}, \mathtt{27})_\lambda, (\mathtt{12}, \mathtt{28})_\lambda, \end{Bmatrix} \tag{7}$$

Every DL distinguisher in this group has a correlation $\mathfrak{c}_j \geq 0.998$ (we have verified them practically). We do not use the DL distinguisher groups in Table 4 because the above one has more masks and higher correlation which leads to a better complexity (using any group of the DL distinguishers in Table 4 also leads to valid attack, with a slightly higher complexity).

Now, we are ready to apply Algorithm 1 to F_{LSL}. It proceeds as follows in each while-loop of Algorithm 1. Set M_0 to be the 128-bit encoding of the integer t. The corresponding X^{-1} can be derived. Under each X^{-1}, we choose one $(\gamma_0, \gamma_1, \delta)$ in $\mathbb{S}_\delta$ and generate $\boldsymbol{y}$ according to Equation (6). Compute the value $\boldsymbol{x} = (x_0, x_1)$ for (X_0^1, X_1^1) from M_0 and $\boldsymbol{y}$. If $\boldsymbol{x} = (x_0, x_1) = (A_{c_0}^{-1}(T_0), A_{c_1}^{-1}(T_1)) = (X_0^2, X_1^2)$, we are done with $(M_0, \boldsymbol{y})$ that can lead to a preimage of (T_0, T_1) according to Equation (4). Otherwise, for $\boldsymbol{y}' = \boldsymbol{y} \oplus (\delta, \delta, 0, 0)$, we test whether $\lambda \cdot (\boldsymbol{x} \oplus (X_0^2, X_1^2)) = \zeta_{\mathfrak{c}_j}$ for all $\lambda_j \in \mathbb{M}$. If $\boldsymbol{y}'$ passes the test, we compute the value $\boldsymbol{x}' = (x_0', x_1')$ for (X_0^2, X_1^2) from M_0 and $\boldsymbol{y}'$. If $\boldsymbol{x}' = (x_0', x_1') = (A_{c_0}^{-1}(T_0), A_{c_1}^{-1}(T_1)) = (X_0^2, X_1^2)$, we can compute the preimage for (T_0, T_1) from $(M_0, \boldsymbol{y}')$ following Equation (4).

Complexity and Success Probability. The size of the output of F_{LSL} is $n = 128$ bits. Since we only use one difference, $s = |\mathbb{D}| = 1$, so $\rho \approx 2^{-0.01}$ and $\tau = 2^{\log(s+1)-n} = 2^{-127}$. The expectation of $|\mathbb{S}_{x,\mathbb{D}}| = \sum_{i=0}^{s-1} 2^{-\ell_i}$ is about 2^{-8}. Thus, to make the success probability of this attack to be about 0.63, we set $N = (\rho\tau)^{-1} = 2^{127.01}$. The time complexity of the attack can be estimated as $N(1 + 2^{-8}) = 2^{127.01} \times (1 + 2^{-8}) \approx 2^{127.02}$ evaluations of F_{LSL}. In our attack, the selection of the $N = 2^{127.01}$ translations can be optimized by randomly choosing, e.g., $2^{117.01}$ M_0 and under each chosen M_0 we traverse all 2^{10} $(\gamma_0, \gamma_1, \delta)$ in $\mathbb{S}_\delta$. With this technique, the computation of M_0 is negligible compared to other parts. Generating $\boldsymbol{y}$ from $(\gamma_0, \gamma_1, \delta)$ costs 4 `Alzette` operations. Further, when pre-computing $(\gamma_0, \gamma_1, \delta)$, we can actually store $(A_{c_0}^{-1}(\gamma_0), A_{c_1}^{-1}(\gamma_1))$. Thus, the cost can be reduced to 2 `Alzette` operations (0.25 steps of `Sparkle`512). Moreover, considering that the nonlinear operations in `XOEsch` is much more costly than the linear layer, we approximately regard the cost of F_{LSL} as that of one step of `Sparkle`512. Thus, to check $\boldsymbol{y}$ costs about 1.25 steps of `Sparkle`512. The 2.5-step `XOEsch`384 instance with a 128-bit digest requires about one 2.5-step `Sparkle`512 (3 nonlinear layers). Consequently, the complexity of the attack is approximately $2^{127.02} \times 1.25/3 \approx 2^{125.76}$ 2.5-step `XOEsch`384 evaluations.

Complexity for the `XOEsch384` with a 192-bit Digest. In the case where the digest is 192-bit (2 blocks), a similar attack as the above one can be mounted. Two blocks of message can provide at most 2^{137} translations of $\hat{\mathbb{D}} = \{0, \delta\}$. We need to use 3-block messages here, denoted by (M_0, M_1, M_2). But still, (M_0, M_1) are computed once for every 2^{10} $(\gamma_0, \gamma_1, \delta)$, using three blocks has no (significant) influence on our complexity. The two digest blocks also have little influence of our attack, except that when our guess matches the first block, we need to continue to match the second block. Since the probability that the first block is matched is very small, the cost for the second matching can be ignored. The final complexity is about $2^{188.76}$ 2.5-step `XOEsch`384 calculations with a 192-bit digest. The successful probability is still about 0.63.

6 Application II: Preimage Attacks on `Ascon-XOF`

`Ascon` has been selected as the NIST LWC standard [DEMS21]. In this section, we give the preimage attacks on the 4-round `Ascon-XOF` with a 128-bit output. In Section K of Supplementary Material, we present preimage attacks on 3-round `Ascon-XOF`. A description of the `Ascon` hash family and its underlying permutation can be found in Section H of Supplementary Material. Note that the last linear layer of `Ascon` permutation can be omitted without affecting our attacks, since an independent and invertible part of the linear layer is applied to the rate which is easy to reverse, and for simplicity, the last linear layer is ignored.

DL Distinguishers for `Ascon-XOF`. The state of the `Ascon` permutation can be represented as four 64-bit words, arranged into 4 rows and 64 columns. The differences and linear masks of the DL distinguishers used in the attacks have only 1 or 2 active bits within the first word. Therefore, we can denote the dif-

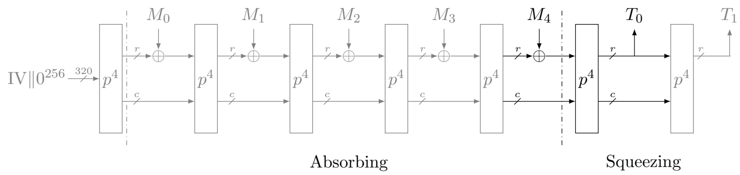


Fig. 5: Illustration of the preimage attack on `Ascon-XOF` with 5 message blocks. The differences are applied to M_4 whereas the masks are put on T_0.

ferences and masks by the column indices of the active bits. For example, (0) means a difference or a linear mask with one active bit located at the 0-th row and 0-th column of the (4×64)-bit state. The DL distinguishers employed in the 4-round attack are produced with $\mathbb{D} = \{\delta_0 = (0), \cdots, \delta_{62} = (62)\}$ and the corresponding $\mathbb{M}_i = \{(i+8), (i+30), (i+50), (i+54)\}, 0 \leq i < 63$. According to the padding rule of `Ascon-XOF`, the message is padded with at least one "1" bit, and thus the last bit of the difference of the messages cannot be active, which is reflected by $(63) \notin \mathbb{D}$. The absolute correlations of the 4-round distinguishers for all $0 \leq i < 63$ are listed as follows:

$$(i) \xrightarrow[0.25]{4R} (i+8), (i) \xrightarrow[0.25]{4R} (i+30), (i) \xrightarrow[0.44]{4R} (i+50), (i) \xrightarrow[0.50]{4R} (i+54).$$

Since $\hat{\mathbb{D}}$ is not a linear space, we have to choose the translations of $\hat{\mathbb{D}}$ in a sufficiently large space to guarantee the disjointness. For `Ascon-XOF` with a 128-bit digest, we need about $2^{128-\log(|\hat{\mathbb{D}}|)} = 2^{122}$ translations. As shown in Figure 5, if we use 5-block messages $(M_0, M_1, M_2, M_3, M_4) \in \mathbb{F}_2^{64\times 5}$ to randomize the selection of the 2^{122} translations, then the probability that they are not disjoint is about $(64^2 \times 2^{244})/2^{321} \approx 2^{-65}$ according to Lemma 2, which is negligible.[4]

Remark 3. All the DL distinguishers of `Ascon` in this work are identified with the method given in [LLL21], and the correlations of all distinguishers have been experimentally verified. When the theoretical correlations differ from the experimental correlations, we take the experimental ones.

Given the 128-bit hash digest $(T_0, T_1) \in \mathbb{F}_2^{64\times 2}$ of `Ascon-XOF`, to recover the preimage $(M_0, M_1, M_2, M_3, M_4)$, we apply Algorithm 1 to the function mapping $(M_0, M_1, M_2, M_3, M_4)$ to (T_0, T_1), where the input differences of the distinguishers are injected through M_4 and the linear masks are applied to T_0. In the attack, we first randomly choose a value for (M_0, M_1, M_2, M_3) and generate the intermediate state X right before the absorbing of M_4. Then, based on X and M_4 we compute the value $x_0 \in \mathbb{F}_2^{64}$ for T_0. If $x_0 = T_0$, we continue to generate x_1 and check if $x_1 = T_1$. If $(x_0, x_1) = (T_0, T_1)$, $(M_0, M_1, M_2, M_3, M_4)$ is then a preimage. Otherwise, for $\delta_i \in \mathbb{D}$, we check if $\lambda \cdot (x_0 \oplus T_0) = \zeta_{\mathfrak{c}_{i,j}}$ holds for all

[4] We can also choose these translations $x \oplus \hat{\mathbb{D}}$ by selecting x only in $\langle \hat{\mathbb{D}} \rangle^{\dashv}$, but this will increase the time complexity by a factor of 2. Because for each while-loop in Algorithm 1, two `Ascon` permutations are evaluated.

$0 \leq j < 4$. If δ_i passes the filter, we use X and $M_4 \oplus \delta_i$ to generate x'_0 and check if $x'_0 = T_0$. If so, we continue to generate x'_1 and check whether $x'_1 = T_1$. If $(x'_0, x'_1) = (T_0, T_1)$, $(M_0, M_1, M_2, M_3, M_4 \oplus \delta_i)$ is a preimage of (T_0, T_1).

Complexity and Success Probability. The output length in this application is $n = 128$. According to our DL distinguishers, the size of the set $\mathbb{D}$ of input differences is $s = |\mathbb{D}| = 63$, so $\rho \approx 2^{-2.16}$ and $\tau = 2^{\log(s+1)-n} = 2^{-122}$. The expectation of $|\mathbb{S}_{x,\mathbb{D}}| = \sum_{i=0}^{s-1} 2^{-4}$ is about $2^{1.98}$. Thus, we let $N = (\rho\tau)^{-1} = 2^{124.16}$ to make the success probability of this attack be 0.63. The time complexity of the attack can be estimated as $N(1 + 2^{1.98}) = 2^{124.16} \times (1 + 2^{1.98}) \approx 2^{126.47}$ evaluations of 4-round `Ascon` permutation.

In our attack, the selection of the $N = 2^{124.16}$ translations can be optimized by randomly choosing, e.g., $2^{104.16}$ (M_0, M_1, M_2, M_3) and under each chosen (M_0, M_1, M_2, M_3) we choose 2^{20} M_4 randomly. With this technique, the computation of (M_0, M_1, M_2, M_3) is negligible compared to other parts. Considering that `Ascon-XOF` with a 128-bit digest requires at least 2 `Ascon` permutations. Our complexity can be scaled to $2^{125.47}$ 4-round `Ascon-XOF` operations. The memory cost is negligible. It is interesting to see that, our preimage attack, as well as the previous two preimage attacks [QHD+23, LHC+23] that can reach 4 rounds of `Ascon-XOF` all have a similar time complexity (the MitM attack additionally costs a significant memory complexity). In addition, by employing the maximum likelihood strategy and the LLR strategy presented in Section 3, we can marginally improve the time complexity of the attack by a factor of $2^{0.06}$, and the details are given in Section I and Section J of the Supplementary Material.

7 Application III: Preimage Attack with State Recovery and MitM

We first apply a similar idea of the preimage attacks in previous sections to recover a particular state of the squeezing phase, then an MitM phase follows to find a proper preimage for the target hash value. As shown in Figure 6, given a hash output $(T_0, T_1, T_2, T_3) \in \mathbb{F}_2^{64\times 4}$, we first recover the capacity part S_{Tc} of S_T by Algorithm 1 with the knowledge of (T_1, T_2, T_3). Then, $S_T = (T_0, S_{Tc})$ can be recovered. Secondly, we enumerate (M_0, M_1) to derive a table $\mathbb{T}_L$ containing 2^{128} states of S_L. Also, with S_T and all possible (M_3, M_4), we derive a table $\mathbb{T}_R$ storing 2^{128} possible S_R. Comparing $\mathbb{T}_L$ and $\mathbb{T}_R$, we can find a pair of (S_L, S_R) that collide in their capacity part (a 256-bit collision). Finally, we compute M_2 from the rate parts of S_L and S_R. The obtained $(M_0, M_1, M_2, M_3, M_4)$ is then a preimage of (T_0, T_1, T_2, T_3). Note that the MitM process can be made memoryless with Floyd's cycle-finding algorithm [Flo67, Sas14].

The designers claimed that `Ascon-HASH` provides 128-bit security with respect to preimage attacks [DEMS21]. However, at CRYPTO 2022 [LM22], Lefevre and Mennink proved that the preimage security bound of a sponge built on an ideal permutation is around $\min\{\max\{n - r', c/2\}, n\}$-bit, where n is the digest size, c the capacity of the sponge (during absorption), and r' the rate (during

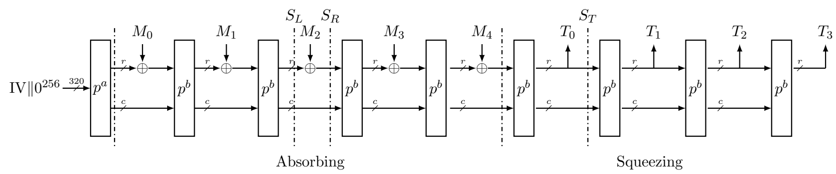


Fig. 6: Illustration of the DL-MitM preimage attack on `Ascon-HASH`.

squeezing). Considering this proof, the preimage security bound of `Ascon-HASH` can be updated to 2^{192} from 2^{128}. In this section, we give a preimage attack on 4-round `Ascon-HASH` with less than 2^{192} `Ascon-HASH` calls. In Section L, we give the 3-round attack.

The most critical step of the above attack is the recovery of S_{Tc}. To speed up the recovery of S_{Tc} with Algorithm 1, the input differences of our DL distinguishers should be active in this part. Thus, if the difference is active in the j-th $(0 \leq j < 64)$ bit of the i-th word $1 \leq i < 5$ of the `Ascon` state, the difference is denoted by $(64 \times i + j)$. Simultaneously, the output mask has active bits in T_1, thus they are denoted by the column index such as $(k), 0 \leq k < 64$. In this way, a DL distinguisher is determined by such a pair of difference and mask. For example, $(64 \times i + j)$ and (k) mean a DL distinguisher for 4-round `Ascon` permutation whose input difference is active in the capacity part of the input and output mask is active in the first word of the output. The absolute correlations of the 4-round distinguishers for $0 \leq i < 64$ are listed as follows

$$(128 + i) \xrightarrow[0.36]{4R} (i + 32), (128 + i) \xrightarrow[0.68]{4R} (i + 54), (128 + i) \xrightarrow[0.24]{4R} (i + 60).$$

Since the 64 differences are all active in the third word of S_T, we can choose disjoint $x \oplus \hat{\mathbb{D}}$ by letting $x \in \langle \hat{\mathbb{D}} \rangle^{\dashv}$. We first guess a $y \in \mathbb{F}_2^{256}$ for the capacity of S_T, together with T_0, we can generate a $x_1 \in \mathbb{F}_2^{64}$ through the 4-round `Ascon` permutation. If $x_1 = T_1$, we continue to generate x_2 and x_3 and check if they match T_2 and T_3, respectively. If all match well, y is a valid capacity of S_T. Otherwise, for $\delta_i \in \mathbb{D}$, we check if $\lambda_{i,j} \cdot (x_1 \oplus T_1) = \zeta_{\mathfrak{c}_{i,j}}$ for $0 \leq j < 3$. If all DL distinguishers hold, we test if $y \oplus \delta_i$ is a valid capacity for S_T.

Complexity and Success Probability. In the recovery of S_T, the output includes 3 blocks whose length is $n = 192$. The size of the differences used is $s = |\mathbb{D}| = 64$. Therefore, $\rho \approx 2^{-1.46}$ and $\tau = 2^{\log(s+1)-n} = 2^{-185.98}$. The expectation of $|\mathbb{S}_{x,\mathbb{D}}| = \sum_{i=0}^{63} 2^{-3}$ is about $2^{2.98}$. Thus, we let $N = (\rho\tau)^{-1} = 2^{187.44}$ to make the success probability be 0.63. The time complexity of the attack can be estimated as $N(1+2^3) = 2^{187.44} \times (1+2^3) \approx 2^{190.61}$ evaluations of 4-round `Ascon` permutation. In the recovery process, only when T_1 is matched, we continue to check if T_2 and T_3 are also matched. Therefore, the computation for T_2 and T_3 are small. Considering that `Ascon-HASH` performs at least 4 permutations, the main part of our calculation is 1/4 of the `Ascon-HASH` computations. Thus, the complexity for recovering S_{Tc} is about $2^{188.61}$.

When a S_T is recovered, we proceed with the internal collision phase. When using 5 messages, the 256-bit internal collision with two 2^{128} sets ($\mathbb{T}_{S_L}$ and $\mathbb{T}_{S_R}$) has a birthday probability. Considering that the collision phase costs around 2^{128} computations, which is negligible compared to the recovery of S_{Tc} phase, we can trade some time and memory with the successful probability. For example, we can use 7 message blocks, and make $\mathbb{T}_{S_L}$ and $\mathbb{T}_{S_R}$ have sizes of, e.g., 2^{130}, then the successful probability of the internal collision phase will boost to extremely close to 1. Further, with Floyd's cycle-finding algorithm [Flo67, Sas14], the internal-collision phase can require a negligible memory cost, so the internal collision phase can be made memoryless. Hence, the time complexity and the success probability are as the same as the recovery of S_{Tc}, which is $2^{188.61}$ and 0.63.

8 Application IV: Key-Recovery Attack on `Schwaemm`

We apply our key-recovery attacks introduced in Section 4 to `Schwaemm`, the AEAD scheme of the `Sparkle` family [BBdS+21]. `Schwaemm` consists of four members, including `Schwaemm`256-256, `Schwaemm`192-192, `Schwaemm`128-128, and `Schwaemm`256-128, where `Schwaemm`r-c instantiates a sponge structure with rate r and capacity c using the permutation $\texttt{Sparkle}(r+c) : \mathbb{F}_2^{r+c} \to \mathbb{F}_2^{r+c}$. Our attack focuses on the initialization phase of the `Schwaemm` family with reduced steps, where the initial state is loaded with r-bit nonce and c-bit key, and processed with the permutation $\texttt{Sparkle}(r+c)$ reduced to R steps. The first r-bit output after the initialization phase is known to the attacker. Our task is to recover the c-bit key. This section shows the attacks on 3.5- and 4.5-step `Schwaemm`256-128. In Section N of Supplementary Material, we give the key-recovery attacks on the other three instances of `Schwaemm` reduced to 3.5 and 4.5 steps.

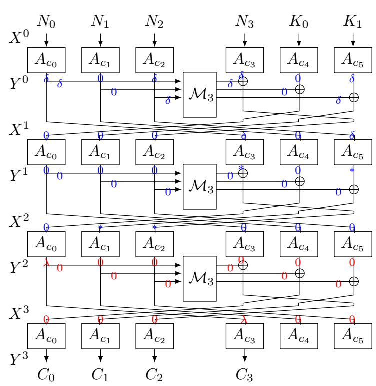


Fig. 7: The initialization phase of `Schwaemm`256-128 reduced to 3.5 steps. The blue values represent the differences whereas the red ones are linear masks.

8.1 Key-Recovery Attack on 3.5-Step Schwaemm256-128

Figure 7 shows the initialization phase of Schwaemm256-128 reduced to 3.5 steps, where $(N_0, N_1, N_2, N_3) \in \mathbb{F}_2^{64\times 4}$ is the 256-bit nonce, $(K_0, K_1) \in \mathbb{F}_2^{64\times 2}$ is the 128-bit key, and $(C_0, C_1, C_2, C_3) \in \mathbb{F}_2^{64\times 4}$ is the 256-bit output known to the attackers. For the convenience of description, the input and output of the j-th step of the Sparkle permutation are denoted by $X^j = (X_0^j, \ldots, X_5^j)$ and $Y^j = (Y_0^j, \ldots, Y_5^j)$, respectively. Under this notation, we have $(X_0^0, X_1^0, X_2^0, X_3^0) = (N_0, N_1, N_2, N_3)$ and $(Y_0^3, Y_1^3, Y_2^3, Y_3^3) = (C_0, C_1, C_2, C_3)$. Given the values of (C_0, C_1, C_2, C_3) and (Y_4^0, Y_5^0), one can obtain the values of $(X_0^3, X_1^3, X_2^3, X_3^3)$ and (K_0, K_1). Therefore, our strategy is to apply Algorithm 5 to the function F_{LSLSL} mapping Y^0 to X^3 to recover (Y_4^0, Y_5^0).

We first introduce the DL distinguishers used for F_{LSLSL}. As shown in Figure 7, let $\Lambda(X^3) = (0, 0, 0, \lambda, 0, 0)$ with $\lambda \in \mathbb{F}_2^{64} \backslash \{0\}$ be the linear mask of X^3. The consequent linear mask of Y^2 is $\Lambda(Y^2) = (\lambda, 0, 0, 0, 0, 0)$. We set the difference of Y^0 to be $\Delta(Y^0) = (\delta, 0, \delta, \delta, 0, \delta)$ with $\delta \in \mathbb{F}_2^{64} \backslash \{0\}$. According to Lemma 3, the difference of X^1 is $\Delta(X^1) = (0, 0, 0, \delta, 0, \delta)$, and thus the difference of X^2 is $\Delta(X^2) = (0, *, *, 0, 0, 0)$, where $*$ can be any nonzero value. Since $\Delta(X_0^2) = 0$, for any nonzero δ and nonzero λ, $\lambda \cdot \Delta(X_3^3) = 0$ holds with certainty. In the application of Algorithm 5 (with necessary tweaks), (Y_4^0, Y_5^0) and $(Y_0^0, Y_1^0, Y_2^0, Y_3^0)$ respectively play the roles of the key and the plaintext, $\mathbb{D} = \{(\delta, 0, \delta, \delta, 0, \delta) : \delta \in \mathbb{F}_2^{64} \backslash \{0\}\}$, $\mathbb{D}_K = \{(0, \delta) : \delta \in \mathbb{F}_2^{64} \backslash \{0\}\}$, $\hat{\mathbb{D}}_K = \{(0, \delta) : \delta \in \mathbb{F}_2^{64}\}$, $\hat{\mathbb{D}}_K^{\dashv} = \{(v, 0) : v \in \mathbb{F}_2^{64}\}$, and the set of masks for all differences in $\mathbb{D}$ can be the same $\mathbb{M} = \{(0, 0, 0, e_i, 0, 0) : 0 \le i < 64\}$, where e_i is the i-th unit vector of $\mathbb{F}_2^{64}$.

In the attack, we randomly choose a value $\boldsymbol{y} = (y_0, y_1, y_2, y_3)$ for $(Y_0^0, Y_1^0, Y_2^0, Y_3^0)$, invert it to obtain the corresponding nonce $\boldsymbol{n} = (n_0, n_1, n_2, n_3)$, and query the Schwaemm256-128 initialization oracle with the $\boldsymbol{n}$ to encrypt a plaintext $\boldsymbol{p}$. From the resulting ciphertext $\boldsymbol{z}$ and $\boldsymbol{p}$, we can deduce the value $\boldsymbol{c} = (c_0, c_1, c_2, c_3)$ for $C = (C_0, C_1, C_2, C_3)$. Inverting $\boldsymbol{c}$ we get $\boldsymbol{x} = (x_0, x_1, x_2, x_3)$ for $(X_0^3, X_1^3, X_2^3, X_3^3)$. Next, for every $\delta \in \mathbb{F}_2^{64} \backslash \{0\}$, we choose $\boldsymbol{y_\delta} = (y_0, y_1, y_2, y_3)_\delta = \boldsymbol{y} \oplus (\delta, 0, \delta, \delta)$ for $(Y_0^0, Y_1^0, Y_2^0, Y_3^0)$, and invert it to obtain $\boldsymbol{n}_\delta$. With the encryption oracle we can get $\boldsymbol{x}_\delta = (x_0, x_1, x_2, x_3)_\delta = (x_{0,\delta}, x_{1,\delta}, x_{2,\delta}, x_{3,\delta})$ for $(X_0^3, X_1^3, X_2^3, X_3^3)$. Then, for each $\boldsymbol{v} = (v, 0) \in \hat{\mathbb{D}}_K^{\dashv}$, we guess the value of (Y_4^0, Y_5^0) to be $\boldsymbol{v}$. Compute $F_{LSLSL}(\boldsymbol{y}, \boldsymbol{v})$, and set $\boldsymbol{w} = (w_0, w_1, w_2, w_3)$ be the first four 64-bit words of $F_{LSLSL}(\boldsymbol{y}, \boldsymbol{v})$. If $\boldsymbol{w} = \boldsymbol{x}$, $\boldsymbol{v}$ is a candidate for (Y_4^0, Y_5^0), and we can confirm its correctness by using additional data. If $\boldsymbol{v}$ is not a candidate for (Y_4^0, Y_5^0) (i.e., $\boldsymbol{w} \neq \boldsymbol{x}$) or $\boldsymbol{v}$ fails to be confirmed as the key, we use the aforementioned DL distinguishers for F_{LSLSL} to quickly filter out those $\boldsymbol{v}_\delta = (v, \delta)$ that cannot be the right value. According to the DL distinguisher, for any nonzero λ, if the difference of Y^0 is $\Delta(Y^0) = (\delta, 0, \delta, \delta, 0, \delta)$, $\lambda \cdot \Delta(X_3^3) = 0$. We have known that w_3 is the result of $(\boldsymbol{y}, \boldsymbol{v}) = (y_0, y_1, y_2, y_3, v, 0)$ and $x_{3,\delta}$ is the result of $(\boldsymbol{y}_\delta, Y_4^0, Y_5^0)$ ($x_{3,\delta}$ is obtained by calling the oracle queried with $\boldsymbol{n}_\delta$). Since $\boldsymbol{y} \oplus \boldsymbol{y}_\delta = (\delta, 0, \delta, \delta)$, if $\boldsymbol{v} \oplus (Y_4^0, Y_5^0) = (v, 0) \oplus (Y_4^0, Y_5^0) = (0, \delta)$, $\lambda \cdot (w_3 \oplus x_{3,\delta}) = 0$ is for sure. Hence, (v, δ) cannot be the right value of (Y_4^0, Y_5^0) if $\lambda \cdot (w_3 \oplus x_{3,\delta}) \neq 0$ for any nonzero λ. Equivalently, only if $\lambda \cdot (w_3 \oplus x_{3,\delta}) = 0$ for all $\lambda \in \mathbb{F}_2^{64} \backslash \{0\}$,

(v, δ) can be a candidate (for a wrong (v, δ), it holds with probability of 2^{-64}, which is the source of the filtering).

Note that $\lambda \cdot (w_3 \oplus x_{3,\delta}) = 0$ for any nonzero λ is equivalent to that $w_3 = x_{3,\delta}$. Therefore, instead of calculating $\lambda \cdot (w_3 \oplus x_{3,\delta})$, we can store all $x_{3,\delta}$ as well as its corresponding $\boldsymbol{y}_\delta$ into a hash table H as $H[x_{3,\delta}] = \boldsymbol{y}_\delta$. Then, we only need to use w_3 to find a collision in H. Note that using this hash table does not increase the memory complexity, because we are always having to store $\boldsymbol{y}_\delta$. When x_3^3 collides with any value in H, we go to confirm $\boldsymbol{v}_\delta = (v, \delta)$ with additional data.

Complexity and Success Probability. To obtain $\boldsymbol{x}$ and $\boldsymbol{x}_\delta$, we need to invert $\boldsymbol{y}$ and $\boldsymbol{y}_\delta$, then call the `Schwaemm`256-128 initialization oracle to obtain $\boldsymbol{c}$ and $\boldsymbol{c}_\delta$, and invert them to $\boldsymbol{x}$ and $\boldsymbol{x}_\delta$, respectively. This process costs about $2^{64} + 2/4 \times 2^{64}$ `Schwaemm`256-128 initialization operations. For each of the 2^{64} $\boldsymbol{v} \in \hat{\mathbb{D}}_K^{\dashv}$, we need to conduct one F_{LSLSL} and one table-lookup. On average, there is one $\boldsymbol{v}_\delta$ that collides with one element of H, so we need another F_{LSLSL} operation to confirm it. Thus, this phase costs about 2^{65} conductions of F_{LSLSL}. Since F_{LSLSL} contains 2 nonlinear layers, its cost can be regarded as 2/4 of the 3.5-step `Schwaemm`256-128 initialization operation. So, the cost of this phase is regarded as 2^{64} `Schwaemm`256-128 initializations. Finally, the whole time complexity is about $2^{64} + 2/4 \times 2^{64} + 2^{64} \approx 2^{65.3}$ `Schwaemm`256-128 initialization operations. The data complexity is 2^{64} nonces. The memory complexity is to store H, which is about 2^{64} 256-bit blocks. Since all DL distinguishers in this application is deterministic, the success probability of recovering it is 1, according to Equation (18).

Remark 4. For `Schwaemm`256-128, there exist structural attacks with comparable complexities. One of these structural attacks is described in Section M in Supplementary Material. This attack was pointed out by one of the reviewers.

8.2 Key-Recovery Attack on 4.5-Step `Schwaemm256-128`

Prepending one round to the 3.5-step attack, we can extend the key-recovery attack to 4.5 steps. Note that choosing any value for $(Y_2^1, Y_3^1, Y_4^1, Y_5^1)$ is possible by controlling the specific nonce value. At the first glance, we can apply Algorithm 5 to the function F_{LSLSL} mapping Y^1 to X^3 to recover (Y_0^1, Y_1^1) then obtain the key. However, the (Y_4^0, Y_5^0) in the 3.5-step attack is a fixed value directly related to (K_0, K_1), on the contrary, (Y_0^1, Y_1^1) here varies according to different $(Y_2^1, Y_3^1, Y_4^1, Y_5^1)$. Hence, our task is to recover (Y_0^1, Y_1^1) that matches the corresponding $(Y_2^1, Y_3^1, Y_4^1, Y_5^1)$.

The DL distinguisher for F_{LSLSL} is the same with the 3.5-step attack except for a step slide. In this attack, it is (Y_0^1, Y_1^1) and $(Y_2^1, Y_3^1, Y_4^1, Y_5^1)$ that respectively play the roles of the key and the plaintext. Thus, the parameters of this attack change accordingly. $\mathbb{D} = \{(\delta, 0, \delta, \delta, 0, \delta) : \delta \in \mathbb{F}_2^{64} \backslash \{0\}\}$, $\mathbb{D}_K = \{(\delta, 0) : \delta \in \mathbb{F}_2^{64} \backslash \{0\}\}$, $\hat{\mathbb{D}}_K = \{(\delta, 0) : \delta \in \mathbb{F}_2^{64}\}$, $\hat{\mathbb{D}}_K^{\dashv} = \{(0, v) : v \in \mathbb{F}_2^{64}\}$, and $\mathbb{M}_i = \{(0, 0, 0, e_i, 0, 0) : 0 \leq i < 64\}$, where e_i is the i-th unit vector of $\mathbb{F}_2^{64}$.

We start by choosing a $\boldsymbol{y} = (y_2, y_3, y_4, y_5)$ for $(Y_2^1, Y_3^1, Y_4^1, Y_5^1)$. See Figure 8, $\boldsymbol{y}$ can be inverted uniquely to $\boldsymbol{n} = (n_0, n_1, n_2, n_3)$. We call the 4.5-step `Schwaemm`256-128 initialization oracle with $\boldsymbol{n}$ to encrypt a plaintext $\boldsymbol{p}$,

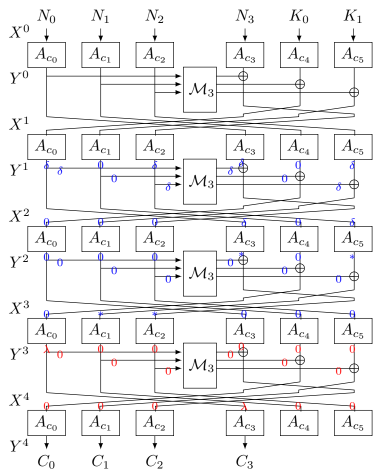


Fig. 8: The illustration of the first steps of the key-recovery attack on 4.5-step Schwaemm256-128 initialization. The underlying permutation is Sparkle384. The blue values represent the differences whereas the red values are masks.

during the process $\boldsymbol{\gamma} = (\gamma_0, \gamma_1)$ is the intermediate value for (Y_0^1, Y_1^1). From the resulting ciphertext $\boldsymbol{z}$ and $\boldsymbol{p}$, we can deduce the value $\boldsymbol{c} = (c_0, c_1, c_2, c_3)$ for $C = (C_0, C_1, C_2, C_3)$. Inverting $\boldsymbol{c}$ we get the value $\boldsymbol{x} = (x_0, x_1, x_2, x_3)$ for $(X_0^4, X_1^4, X_2^4, X_3^4)$. Next, for every $\delta \in \mathbb{F}_2^{64} \backslash \{0\}$, we choose $\boldsymbol{y}_\delta = \boldsymbol{y} \oplus (\delta, \delta, 0, \delta)$ for $(Y_2^1, Y_3^1, Y_4^1, Y_5^1)$, and invert it to obtain $\boldsymbol{n}_\delta$. With the encryption oracle we can get $\boldsymbol{x}_\delta = (x_0, x_1, x_2, x_3)_\delta = (x_{0,\delta}, x_{1,\delta}, x_{2,\delta}, x_{3,\delta})$ for $(X_0^4, X_1^4, X_2^4, X_3^4)$, where $\boldsymbol{\gamma}_\delta = (\gamma_{0,\delta}, \gamma_{1,\delta})$ is the intermediate value for (Y_0^1, Y_1^1).

Similar to the 3.5-step attack, for each $\boldsymbol{v} = (0, v) \in \mathbb{F}_2^{64}$, we guess the value (γ_0, γ_1) to be $\boldsymbol{v}$. Compute $F_{LSLSL}(\boldsymbol{v}, \boldsymbol{y})$, and set $\boldsymbol{w} = (w_0, w_1, w_2, w_3)$ be the first four 64-bit words of $F_{LSLSL}(\boldsymbol{v}, \boldsymbol{y})$. If $\boldsymbol{w} = \boldsymbol{x}$, $\boldsymbol{v}$ is a candidate for $\boldsymbol{\gamma}$. Then, we invert $(\boldsymbol{v}, \boldsymbol{y})$ to get $(\boldsymbol{n}, \boldsymbol{k})$ and use $\boldsymbol{k}$ to test other $\boldsymbol{n}_\delta$ and $\boldsymbol{x}_\delta$ to confirm $\boldsymbol{k}$. If $\boldsymbol{v}$ is not the candidate (i.e., $\boldsymbol{w} \neq \boldsymbol{x}$) or $\boldsymbol{k}$ fails to be confirmed as the key. We use the DL distinguishers of F_{LSLSLS} to quickly filter those $\boldsymbol{v}_\delta = (\delta, v)$ that cannot be a candidate. According to the DL distinguisher, for any $\lambda \neq 0$, if $\Delta(Y^1) = (\delta, 0, \delta, \delta, 0, \delta)$, $\lambda \cdot \Delta(X_3^4) = 0$. The state Y^1 for w_3 is $(\boldsymbol{v}, \boldsymbol{y})$, whereas the state Y^1 for $x_{3,\delta}$ is $(\boldsymbol{\gamma}_\delta, \boldsymbol{y}_\delta)$. Since $\boldsymbol{y} \oplus \boldsymbol{y}_\delta = (\delta, \delta, 0, \delta)$, if $\boldsymbol{v} \oplus \boldsymbol{\gamma}_\delta = (0, v) \oplus (\gamma_{0,\delta}, \gamma_{1,\delta}) = (\delta, 0)$, $\lambda \cdot (w_3 \oplus x_{3,\delta}) = 0$ occurs for any nonzero λ and $(0, v) \oplus (\delta, 0) = (\delta, v)$ will be regarded as a candidate for $\boldsymbol{\gamma}_\delta = (\gamma_{0,\delta}, \gamma_{1,\delta})$. To detect $\lambda \cdot (w_3 \oplus x_{3,\delta}) = 0$, we can also use a hash table as we did in the 3.5-step attack to quickly find the collision between w_3 and $x_{3,\delta}$. If $\boldsymbol{\gamma}_\delta = (\delta, v)$ for some δ and v, it can be detected and then confirmed by other data.

Complexity and Success Probability. The 4.5-step attack requires 2^{64} chosen nonces. To prepare these nonces, we need to first invert all of them from $(Y_2^1, Y_3^1, Y_4^1, Y_5^1)$ to the nonce, then call the Schwaemm256-128 initialization oracle to handle all the nonces to derive the corresponding outputs. After that, we invert the output back through one nonlinear layer. This process costs $2^{64} + 3/5 \times 2^{64}$ 4.5-steps initializations. When recovering (Y_0^1, Y_1^1), it fully follows Algorithm 5, which mainly costs about 2^{65} F_{LSLSL} operations for all the translations, which is equivalent to $2/5 \times 2^{65}$ the `Schwaemm`256-128 initializations. Thus, the final time complexity is about $2^{65.4}$ `Schwaemm`256-128 initializations. The memory complexity is also for storing the hash table, which is about 2^{64} 256-bit blocks. Different from the 3.5-step attack where the only key can always be recovered, $\boldsymbol{\gamma}_\delta$ (include $\boldsymbol{\gamma}$ that can be seen $\boldsymbol{\gamma}_\delta$ with $\delta = 0$) is not necessarily hit by Algorithm 5. Only when $\boldsymbol{\gamma}_\delta = (\delta, v) = (0, v) \oplus (\delta, 0)$ for at least one $v \in \mathbb{F}_2^{64}$, it can be hit and recovered. Assume that the mapping sending (K_0, K_1) to (Y_0^1, Y_1^1) is a random function. For a specific $v \in \mathbb{F}_2^{64}$, the probability that $\boldsymbol{\gamma}_\delta$ is hit by (δ, v) is approximately 2^{-64}; for all $v \in \mathbb{F}_2^{64}$, the probability that at least one $v \in \mathbb{F}_2^{64}$ makes $\boldsymbol{\gamma}_\delta$ be hit is about $1 - (1 - 2^{-64})^{2^{64}} \approx 1 - e^{-1} \approx 0.63$.

Remark 5. In the specification [BBdS+21], a data limit for `Schwaemm`256-128 was set to be 2^{68} bytes, i.e., 2^{63} 256-bit blocks. The authors wrote, "*The data limits correspond to* 2^{64} *blocks of* r *bits rounded up to the closest power of two* ...". Thus, the data limit should be 2^{64} 256-bit blocks for `Schwaemm`256-128. Our attack costs 2^{64} 256-bit blocks, which is valid considering the latter data limit.

9 Conclusion

This work shows that the preimage and key-recovery attacks can be accelerated in a generic way whenever a proper set of highly biased differential-linear distinguishers are identified for the targeted (parameterized) one-way function. The technique is quite versatile as demonstrated by the applications. From these applications, we see that it is possible to exploit related-key differential-linear distinguishers in the single-key model without querying the encryption oracle with unknown but related-keys. This evidence the importance of security analysis in the related-key model, and alert the designers in designing primitives meant to be secure only in the single-key model without thorough related-key cryptanalysis. On the other hand, the limitation of the method is that it relies on extremely strong distinguishers and it is exhaustive search in nature. We believe that this technique will find more applications in the future.

Acknowledgment. We thank the reviewers for their valuable comments. This work is supported by National Key Research and Development Program of China (2022YFB2701900, 2018YFA0704702), the Natural Science Foundation of China (62032014, U2336207), and the Fundamental Research Funds for the Central Universities. Kai Hu and Meiqin Wang are supported by the Major Basic Research Project of Natural Science Foundation of Shandong Province, China (Grant No. ZR202010220025), Department of Science & Technology of Shandong Province (No.SYS202201), Quan Cheng Laboratory (Grant No. QCLZD202301, QCLZD202306). Finally, Kai Hu is also supported by the Program of Qilu Young Scholars of Shandong University, the France-Singapore NRF-ANR research grant NRF2020-NRF-ANR072, the Singapore NRF Investigatorship research grant NRF-NRFI08-2022-0013.

References

ASC. Lightweight Cryptography Standardization Process: NIST Selects `Ascon`. https://csrc.nist.gov/News/2023/lightweight-cryptography-nist-selects-ascon.

BBC+22. Christof Beierle, Marek Broll, Federico Canale, Nicolas David, Antonio Flórez-Gutiérrez, Gregor Leander, María Naya-Plasencia, and Yosuke Todo. Improved Differential-Linear Attacks with Applications to ARX Ciphers. *J. Cryptol.*, 35(4):29, 2022.

BBdS+20. Christof Beierle, Alex Biryukov, Luan Cardoso dos Santos, Johann Großschädl, Léo Perrin, Aleksei Udovenko, Vesselin Velichkov, and Qingju Wang. `Alzette`: A 64-Bit ARX-box - (Feat. `Crax` and `Trax`). In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology - CRYPTO 2020*, volume 12172 of *LNCS*, pages 419–448. Springer, 2020.

BBdS+21. Christof Beierle, Alex Biryukov, Luan Cardoso dos Santos, Johann Großschädl, Léo Perrin, Aleksei Udovenko, Vesselin Velichkov, Qingju Wang, and Alex Biryukov. `Schwaemm` and `Esch`: lightweight authenticated encryption and hashing using the Sparkle permutation family. *NIST round*, 3, 2021.

BCG+12. Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knezevic, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçin. `Prince` - A Low-Latency Block Cipher for Pervasive Computing Applications, - Extended Abstract. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology - ASIACRYPT 2012*, volume 7658 of *Lecture Notes in Computer Science*, pages 208–225. Springer, 2012.

BCP22. Jules Baudrin, Anne Canteaut, and Léo Perrin. Practical Cube Attack against Nonce-Misused `Ascon`. *IACR Trans. Symmetric Cryptol.*, 2022(4):120–144, 2022.

BDKW19. Achiya Bar-On, Orr Dunkelman, Nathan Keller, and Ariel Weizman. DLCT: A New Tool for Differential-Linear Cryptanalysis. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2019*, volume 11476 of *LNCS*, pages 313–342. Springer, 2019.

BDP+18. Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche, Ronny Van Keer, and Benoît Viguier. KangarooTwelve: Fast Hashing Based on `Keccak`-p. In Bart Preneel and Frederik Vercauteren, editors, *Applied Cryptography and Network Security - 16th International Conference, ACNS 2018*, volume 10892 of *LNCS*, pages 400–418. Springer, 2018.

BDPA08. Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. On the Indifferentiability of the Sponge Construction. In Nigel P. Smart, editor, *Advances in Cryptology - EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 181–197. Springer, 2008.

BDPA11. Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Duplexing the Sponge: Single-Pass Authenticated Encryption and Other, applications. In Ali Miri and Serge Vaudenay, editors, *Selected Areas in Cryptography - SAC 2011*, volume 7118 of *LNCS*, pages 320–337. Springer, 2011.

BDPA13. Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. `Keccak`. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology - EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 313–314. Springer, 2013.

BDPVA07. Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Sponge functions. In *ECRYPT hash workshop*, volume 2007, 2007.

BKR11. Andrey Bogdanov, Dmitry Khovratovich, and Christian Rechberger. Biclique cryptanalysis of the full AES. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology - ASIACRYPT 2011*, volume 7073 of *Lecture Notes in Computer Science*, pages 344–371. Springer, 2011.

Dae91. Joan Daemen. Limitations of the Even-Mansour Construction. In Hideki Imai, Ronald L. Rivest, and Tsutomu Matsumoto, editors, *Advances in Cryptology - ASIACRYPT 1991*, volume 739 of *LNCS*, pages 495–498. Springer, 1991.

DEMS15. Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Cryptanalysis of `Ascon`. In Kaisa Nyberg, editor, *Topics in Cryptology - CT-RSA 2015*, volume 9048 of *LNCS*, pages 371–387. Springer, 2015.

DEMS21. Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. `Ascon` v1.2: Lightweight Authenticated Encryption and Hashing. *J. Cryptol.*, 34(3):33, 2021.

DKS12. Orr Dunkelman, Nathan Keller, and Adi Shamir. Minimalism in Cryptography: The Even-Mansour Scheme Revisited. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology - EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 336–354. Springer, 2012.

DMP+15. Itai Dinur, Pawel Morawiecki, Josef Pieprzyk, Marian Srebrny, and Michal Straus. Cube Attacks and Cube-Attack-Like Cryptanalysis on the Round-Reduced, keccak sponge function. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015*, volume 9056 of *Lecture Notes in Computer Science*, pages 733–761. Springer, 2015.

Flo67. Robert W. Floyd. Nondeterministic Algorithms. *J. ACM*, 14(4):636–644, 1967.

GLS16. Jian Guo, Meicheng Liu, and Ling Song. Linear Structures: Applications to Cryptanalysis of Round-Reduced, keccak. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016*, volume 10031 of *Lecture Notes in Computer Science*, pages 249–274, 2016.

GPT21. David Gérault, Thomas Peyrin, and Quan Quan Tan. Exploring Differential-Based Distinguishers and Forgeries for `Ascon`. *IACR Trans. Symmetric Cryptol.*, 2021(3):102–136, 2021.

HCN08. Miia Hermelin, Joo Yeon Cho, and Kaisa Nyberg. Multidimensional linear cryptanalysis of reduced round Serpent. In Yi Mu, Willy Susilo, and Jennifer Seberry, editors, *Information Security and Privacy, 13th Australasian Conference, ACISP 2008*, volume 5107 of *LNCS*, pages 203–215. Springer, 2008.

HLY21. Le He, Xiaoen Lin, and Hongbo Yu. Improved Preimage Attacks on 4-Round `Keccak`-224/256. *IACR Trans. Symmetric Cryptol.*, 2021(1):217–238, 2021.

HMS+76. Martin Hellman, Ralph Merkle, Richard Schroeppel, Lawrence Washington, Whitfield Diffie, and P Schweitzer. *Results of an initial attempt to cryptanalyze the NBS Data Encryption Standard.* 1976.

HWX+17. Senyang Huang, Xiaoyun Wang, Guangwu Xu, Meiqin Wang, and Jingyuan Zhao. Conditional Cube Attack on Reduced-Round `Keccak` Sponge Function. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology - EUROCRYPT 2017*, volume 10211 of *Lecture Notes in Computer Science*, pages 259–288, 2017.

LH94. Susan K. Langford and Martin E. Hellman. Differential-Linear Cryptanalysis. In Yvo Desmedt, editor, *Advances in Cryptology - CRYPTO '94*, volume 839 of *LNCS*, pages 17–25. Springer, 1994.

LHC+23. Huina Li, Le He, Shiyao Chen, Jian Guo, and Weidong Qiu. Automatic Preimage Attack Framework on `Ascon` Using a Linearize-and-Guess Approach. *Cryptology ePrint Archive*, 2023.

LIMY21. Fukang Liu, Takanori Isobe, Willi Meier, and Zhonghao Yang. Algebraic Attacks on Round-Reduced `Keccak`. In Joonsang Baek and Sushmita Ruj, editors, *Information Security and Privacy - 26th Australasian Conference, ACISP 2021*, volume 13083 of *LNCS*, pages 91–110. Springer, 2021.

LLL21. Meicheng Liu, Xiaojuan Lu, and Dongdai Lin. Differential-Linear Cryptanalysis from an Algebraic Perspective. In *Advances in Cryptology - CRYPTO 2021*, volume 12827 of *LNCS*, pages 247–277. Springer, 2021.

LM22. Charlotte Lefevre and Bart Mennink. Tight Preimage Resistance of the Sponge Construction. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology - CRYPTO 2022*, volume 13510 of *Lecture Notes in Computer Science*, pages 185–204. Springer, 2022.

LS19. Ting Li and Yao Sun. Preimage Attacks on Round-Reduced `Keccak`-224/256 via an Allocating, approach. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2019*, volume 11478 of *Lecture Notes in Computer Science*, pages 556–584. Springer, 2019.

LSLW17. Ting Li, Yao Sun, Maodong Liao, and Dingkang Wang. Preimage Attacks on the Round-reduced `Keccak` with Cross-linear Structures. *IACR Trans. Symmetric Cryptol.*, 2017(4):39–57, 2017.

Lu15. Jiqiang Lu. A methodology for differential-linear cryptanalysis and its applications. *Des. Codes Cryptogr.*, 77(1):11–48, 2015.

NSLL22. Zhongfeng Niu, Siwei Sun, Yunwen Liu, and Chao Li. Rotational Differential-Linear Distinguishers of ARX Ciphers with Arbitrary Output Linear Masks. In *Advances in Cryptology - CRYPTO 2022*, volume 13507 of *LNCS*, pages 3–32. Springer, 2022.

QHD+23. Lingyue Qin, Jialiang Hua, Xiaoyang Dong, Hailun Yan, and Xiaoyun Wang. Meet-in-the-Middle Preimage Attacks on Sponge-Based Hashing. In *Advances in Cryptology - EUROCRYPT 2023*, volume 14007 of *LNCS*, pages 158–188. Springer, 2023.

QZH+23. Lingyue Qin, Boxin Zhao, Jialiang Hua, Xiaoyang Dong, and Xiaoyun Wang. Weak-Diffusion Structure: Meet-in-the-Middle Attacks on Sponge-based, Hashing Revisited. *IACR Cryptol. ePrint Arch.*, page 518, 2023.

Sas14. Yu Sasaki. Memoryless Unbalanced Meet-in-the-Middle Attacks: Impossible Results, and applications. In Ioana Boureanu, Philippe Owesarski, and Serge Vaudenay, editors, *Applied Cryptography and Network Security - 12th International Conference, ACNS 2014*, volume 8479 of *Lecture Notes in Computer Science*, pages 253–270. Springer, 2014.

SS22. André Schrottenloher and Marc Stevens. Simplified MITM Modeling for Permutations: New (Quantum) Attacks. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology - CRYPTO 2022*, volume 13509 of *LNCS*, pages 717–747. Springer, 2022.

TMC+23. Meltem Sönmez Turan, Kerry McKay, Donghoon Chang, Jinkeon Kang, Noah Waller, John M Kelsey, Lawrence E Bassham, and Deukjo Hong. Status report on the final round of the NIST lightweight cryptography standardization process. 2023.

ZDW19. Rui Zong, Xiaoyang Dong, and Xiaoyun Wang. Collision Attacks on Round-Reduced Gimli-Hash/`Ascon-XOF`/`Ascon-HASH`. *IACR Cryptol. ePrint Arch.*, page 1115, 2019.

Supplementary Material

A Proof of Lemma 2

Proof. For two randomly chosen x_i and x_j, the event that there is at least one element of $x_i \oplus \hat{\mathbb{D}}$ appearing in $x_j \oplus \hat{\mathbb{D}}$ is denoted as $e_{i,j}$. Then the event $e_{i,j}$ means that there exits an element $\gamma = \gamma_1 \oplus \gamma_2$ where $\gamma_1, \gamma_2 \in \hat{\mathbb{D}}$ such that $x_i = x_j \oplus \gamma$. Thus, for some $a \in \mathbb{F}_2^m$, we have

$$\Pr[e_{i,j} | x_j = a] \leq \frac{(s+1)^2}{2^m}.$$

Then, we can see that

$$\Pr[e_{i,j}] = \sum_{a \in \mathbb{F}_2^m} \Pr[e_{i,j} | x_j = a] \Pr[x_j = a] = \frac{1}{2^m} \sum_{a \in \mathbb{F}_2^m} \Pr[e_{i,j} | x_j = a] \leq \frac{(s+1)^2}{2^m}$$

We denote the event that any two translation of $\hat{\mathbb{D}}$ among

$$x_0 \oplus \hat{\mathbb{D}}, x_1 \oplus \hat{\mathbb{D}}, \ldots, x_{\alpha-1} \oplus \hat{\mathbb{D}}$$

share a common value by A. Then, $A = \bigcup_{0 \leq i,j < \alpha} e_{i,j}$. Therefore, we have

$$\Pr[A] = \Pr\left[\bigcup_{0 \leq i,j < \alpha} e_{i,j}\right] \leq \sum_{0 \leq i,j < \alpha} \Pr[e_{i,j}] \leq \frac{(s+1)^2 \alpha(\alpha-1)}{2^{m+1}}.$$

□

B Proofs of Lemmas 4 and 5

The following is the proof of Lemma 4:

Proof. Let $z_i = (x_i, y_i) \in \mathbb{F}_2^{32 \times 2}$ $(0 \leq i < w)$ and $I_\lambda = \{0 \leq i < w : \gamma_i = \lambda\}$. Then

$$\text{LHS} = \gamma \cdot (z_0, z_1, z_2, z_3, \ldots, z_{w-1}) = \lambda \cdot \bigoplus_{i \in I_\lambda} (x_i, y_i).$$

Simultaneously, $z_i' = (x_i \oplus \ell(t_y), y_i \oplus \ell(t_x))$, where t_x and t_y are calculated according to Definition 1. Therefore,

$$\text{RHS} = \gamma \cdot \left(z_0', z_1', z_2', z_3', \ldots, z_{w-1}'\right) = \lambda \cdot \bigoplus_{i \in I_\lambda} \left(x_i \oplus \ell(t_y), y_i \oplus \ell(t_x)\right),$$

when $|I_\lambda|$ is even, all $\ell(t_x)$ and $\ell(t_y)$ are canceled. Thus LHS = RHS, which ends the proof. □

The following is the proof of Lemma 5:

Proof. According to Definition 1, we have the follow equations:

$$\begin{cases} (u_0, v_0) = (x_0 \oplus \ell(y_0 \oplus y_1), y_0 \oplus \ell(x_0 \oplus x_1)) \\ (u_1, v_1) = (x_1 \oplus \ell(y_0 \oplus y_1), y_1 \oplus \ell(x_0 \oplus x_1)) \end{cases}$$

Thus, $u_0 \oplus u_1 = x_0 \oplus x_1$ and $v_0 \oplus v_1 = y_0 \oplus y_1$. Replace $x_0 \oplus x_1$, $y_0 \oplus y_1$ with $u_0 \oplus u_1$ and $v_0 \oplus v_1$, respectively, we derive

$$\begin{cases} (x_0, y_0) = (u_0 \oplus \ell(v_0 \oplus v_1), v_0 \oplus \ell(u_0 \oplus u_1)) \\ (x_1, y_1) = (u_1 \oplus \ell(v_0 \oplus v_1), v_1 \oplus \ell(u_0 \oplus u_1)) \end{cases}$$

Thus, any $((u_0, v_1), (u_1, v_1))$ can lead to a unique $((x_0, y_0), (x_1, y_1))$.

In addition, according to Definition 1 again, for $2 \leq i < w$,

$$(u_i, v_i) = (\ell(y_0 \oplus y_1), \ell(x_0 \oplus x_1)) = (\ell(v_0 \oplus v_1), \ell(u_0 \oplus u_1)) \,.$$

□

C More Advanced Statistical Tests

C.1 The Analysis of the Threshold Strategy Given in Algorithm 3

Let $\mathbb{D} = \{\delta_0, \delta_1, \ldots, \delta_{s-1}\} \subseteq \mathbb{F}_2^m$ be a set of s *nonzero* differences. For each δ_i $(0 \leq i < s)$, there is a set $\mathbb{M}_i = \{\lambda_{i,0}, \lambda_{i,1}, \ldots, \lambda_{i,\ell_i - 1}\}$ of ℓ_i linearly-independent linear masks, such that each $(\delta_i, \lambda_{i,j})$ forms a DL distinguisher with correlation $\mathfrak{c}_{i,j}$. For $\delta_i \in \mathbb{D}$ and $0 \leq j < \ell_i$, let $w_{i,j} = \zeta_{\mathfrak{c}_{i,j}} \oplus \lambda_{i,j} \cdot (y \oplus O)$, then

$$num = \ell_i - \sum_{j=0}^{\ell_i - 1} w_{i,j}.$$

In Algorithm 3, an element in a translation is accepted only when the num is at least γ_i. Let p_i be probability that the num is at least γ_i when $F(x \oplus \delta_i) = O$ and q_i be probability that the num is at least γ_i when $F(x \oplus \delta_i) \neq O$. Then,

$$p_i = \sum_{\substack{u \in \mathbb{F}_2^{\ell_i}, \\ wt(u) < \gamma_i}} \prod_{j=0}^{\ell_i - 1} \left(\frac{1}{2} + \frac{(-1)^{u_i} \mathfrak{c}_{i,j}}{2} \right) \quad \text{and} \quad q_i = \sum_{z=\gamma_i}^{\ell_i} \binom{\ell_i}{z} 2^{-\ell_i}$$

Complexity Analysis. When `PreTest()` is instantiated with Algorithm 3, $\mathbb{S}_{x,\mathbb{D}} = \{x \oplus \delta_i : \delta_i \in \mathbb{D}, num \geq \gamma_i\}$. On average, we expect $|\mathbb{S}_{x,\mathbb{D}}| = \sum_{i=0}^{s-1} q_i$ for a random x. Consequently, the complexity of Algorithm 1 is about $N \left(1 + \sum_{i=0}^{s-1} q_i\right)$ evaluations of F. Generally, the complexity of the inner products is negligible compared with the complexity due to the evaluations of F.

Success Probability. The probability q of hitting a preimage in one while-loop of Algorithm 1 with a random guess $x \in \mathbb{F}_2^m$ can be computed as

$$q \geq \Pr[F(x) = O] + \sum_{i=0}^{s-1} \Pr[F(x \oplus \delta_i) = O \text{ and } x \oplus \delta_i \in \mathbb{S}_{x,\mathbb{D}}]. \tag{8}$$

For $0 \leq i < s$, we have

$$\begin{aligned} & \Pr[F(x \oplus \delta_i) = O \text{ and } x \oplus \delta_i \in \mathbb{S}_{x,\mathbb{D}}] \\ = \; & \Pr[x \oplus \delta_i \in \mathbb{S}_{x,\mathbb{D}} \mid F(x \oplus \delta_i) = O] \; \Pr[F(x \oplus \delta_i) = O] \\ = \; & \Pr[x \oplus \delta_i \in \mathbb{S}_{x,\mathbb{D}} \mid F(x \oplus \delta_i) = O] \left(1 - \frac{1}{2^n}\right)^{i+1} \frac{1}{2^n} \\ = \; & p_i \left(1 - \frac{1}{2^n}\right)^{i+1} \frac{1}{2^n} > p_i \left(1 - \frac{1}{2^n}\right)^{s} \frac{1}{2^n}, \end{aligned} \tag{9}$$

where

$$p_i = \sum_{\substack{u \in \mathbb{F}_2^{\ell_i}, \\ wt(u) < \gamma_i}} \prod_{j=0}^{\ell_i - 1} \left(\frac{1}{2} + \frac{(-1)^{u_i} \mathfrak{c}_{i,j}}{2} \right).$$

Substituting Equation (9) into Equation (8) gives

$$q > \frac{1}{2^n} + \sum_{i=0}^{s-1} p_i \left(1 - \frac{1}{2^n}\right)^{s} \frac{1}{2^n}.$$

Since $s \ll 2^n$ and $\left(1 - \frac{1}{2^n}\right)^s = \left(1 - \frac{1}{2^n}\right)^{2^n \frac{s}{2^n}} \approx e^{-\frac{s}{2^n}} \approx 1$, we have

$$q > \frac{1}{2^n} + \sum_{i=0}^{s-1} \frac{p_i}{2^n} = 2^{\log(s+1)-n} \frac{1}{s+1} \left(1 + \sum_{i=0}^{s-1} p_i\right) = \rho\tau,$$

where $\tau = 2^{\log(s+1)-n}$ and $\rho = \frac{1}{s+1}(1 + \sum_{i=0}^{s-1} p_i)$. Therefore, the success probability that a preimage is detected after N while-loops of Algorithm 1 is lower bounded by $\mathsf{P}_{suc} = 1 - (1 - \rho\tau)^N$. In this work, we always set $N = (\rho\tau)^{-1}$ to make the success probability to be about $1 - e^{-1} \approx 0.63$.

C.2 The Analysis of the Maximum Likelihood Strategy

Let $\mathbb{D} = \{\delta_0, \delta_1, \ldots, \delta_{s-1}\} \subseteq \mathbb{F}_2^m$ be a set of s *nonzero* differences. For each $\delta_i \in \mathbb{D}$, there is a set $\mathbb{M}_i = \{\lambda_{i,0}, \lambda_{i,1}, \ldots, \lambda_{i,\ell_i - 1}\}$ of ℓ_i linearly independent linear masks, such that each $(\delta_i, \lambda_{i,j})$ forms a DL distinguisher with correlation $\mathfrak{c}_{i,j}$. We define $L^{(i)} : \mathbb{F}_2^m \mapsto \mathbb{F}_2^{\ell_i}$ to be the function mapping $x \in \mathbb{F}_2^m$ to

$$(\lambda_{i,0} \cdot (F(x) \oplus F(x \oplus \delta_i)), \cdots, \lambda_{i,\ell_i - 1} \cdot (F(x) \oplus F(x \oplus \delta_i)))$$

For $u \in \mathbb{F}_2^{\ell_i}$, let $g_u^{(i)} = \Pr_{x \in \mathbb{F}_2^m}[L^{(i)}(x) = u]$ and

$$\mathcal{N}_{\gamma_i} = \{u \in \mathbb{F}_2^{\ell_i} : g_u^{(i)} \geq \gamma_i\}.$$

For $g_u^{(i)}$, according to [Lu15, HCN08], we have

$$\begin{aligned} g_u^{(i)} &= \sum_{x \in \mathbb{F}_2^m} \theta(u \oplus L^{(i)}(x)) \\ &= \frac{1}{2^{\ell_i}} \sum_{a \in \mathbb{F}_2^{\ell_i}} (-1)^{a \cdot u} \sum_{x \in \mathbb{F}_2^m} (-1)^{a \cdot L^{(i)}(x)} \\ &= \frac{1}{2^{\ell_i}} \sum_{a=(a_0,\cdots,a_{\ell_i-1}) \in \mathbb{F}_2^{\ell_i}} (-1)^{a \cdot u} \left(\sum_{x \in \mathbb{F}_2^m} (-1)^{\left(\left(\sum_{j=0}^{\ell_i} a_j \lambda_{i,j}\right) \cdot (F(x) \oplus F(x \oplus \delta_i))\right)} \right) \end{aligned}$$

where $\theta : \mathbb{F}_2^{\ell_i} \to \{0, 1\}$ such that

$$\theta(v) = \begin{cases} 1, & v = 0 \\ 0, & \text{otherwise} \end{cases}.$$

Then, we can implement `PreTest()` with the maximum likelihood strategy as shown in Algorithm 4.

Complexity Analysis. When `PreTest()` is instantiated with Algorithm 4, $\mathbb{S}_{x,\mathbb{D}} = \{x \oplus \delta_i : \delta_i \in \mathbb{D}, (\lambda_{i,0} \cdot (F(x) \oplus O), \cdots, \lambda_{i,\ell_i-1} \cdot (F(x) \oplus O) \in \mathcal{N}_{\gamma_i}\}$. Thus, on average we expect $|\mathbb{S}_{x,\mathbb{D}}| = \sum_{i=0}^{s-1} \frac{|\mathcal{N}_{\gamma_i}|}{2^{\ell_i}}$ for a random x. Consequently, the complexity of Algorithm 1 is about $N\left(1 + \sum_{i=0}^{s-1} \frac{|\mathcal{N}_{\gamma_i}|}{2^{\ell_i}}\right)$ evaluations of F.

Success Probability. The probability q of hitting a preimage in one while-loop of Algorithm 1 with a random guess $x \in \mathbb{F}_2^m$ can be computed as

$$q \geq \Pr[F(x) = O] + \sum_{i=0}^{s-1} \Pr[F(x \oplus \delta_i) = O \text{ and } x \oplus \delta_i \in \mathbb{S}_{x,\mathbb{D}}]. \tag{10}$$

For $0 \leq i < s$, we have

$$\begin{aligned} & \Pr[F(x \oplus \delta_i) = O \text{ and } x \oplus \delta_i \in \mathbb{S}_{x,\mathbb{D}}] \\ = \ & \Pr[x \oplus \delta_i \in \mathbb{S}_{x,\mathbb{D}} \mid F(x \oplus \delta_i) = O] \ \Pr[F(x \oplus \delta_i) = O] \\ = \ & \Pr[x \oplus \delta_i \in \mathbb{S}_{x,\mathbb{D}} \mid F(x \oplus \delta_i) = O] \left(1 - \frac{1}{2^n}\right)^{i+1} \frac{1}{2^n} \\ = \ & p_i \left(1 - \frac{1}{2^n}\right)^{i+1} \frac{1}{2^n} > p_i \left(1 - \frac{1}{2^n}\right)^{s} \frac{1}{2^n}, \end{aligned} \tag{11}$$

where $p_i = \sum_{u \in \mathcal{N}_{\gamma_i}} g_u^{(i)}$. If the differential-linear approximations $(\delta_i, \lambda_{i,j})$, $0 \le j < \ell_i$ of F with correlation $\mathfrak{c}_{i,j}$ are independent with each other, for $u = (u_0, \cdots, u_{\ell_i - 1}) \in \mathbb{F}_2^{\ell_i}$,

$$g_u^{(i)} = \prod_{j=0}^{\ell_i - 1} \left(\frac{1}{2} + \frac{(-1)^{u_i} \mathfrak{c}_{i,j}}{2} \right). \tag{12}$$

Substituting Equation (11) into Equation (10) gives

$$q > \frac{1}{2^n} + \sum_{i=0}^{s-1} p_i \left(1 - \frac{1}{2^n}\right)^s \frac{1}{2^n}.$$

Since $s \ll 2^n$ and $\left(1 - \frac{1}{2^n}\right)^s = \left(1 - \frac{1}{2^n}\right)^{2^n \frac{s}{2^n}} \approx e^{-\frac{s}{2^n}} \approx 1$, we have

$$q > \frac{1}{2^n} + \sum_{i=0}^{s-1} \frac{p_i}{2^n} = 2^{\log(s+1)-n} \frac{1}{s+1} \left(1 + \sum_{i=0}^{s-1} p_i\right) = \rho\tau,$$

where $\tau = 2^{\log(s+1)-n}$ and $\rho = \frac{1}{s+1}(1 + \sum_{i=0}^{s-1} p_i)$. Therefore, the success probability that a preimage is detected after N while-loops of Algorithm 1 is lower bounded by $\mathsf{P}_{suc} = 1 - (1 - \rho\tau)^N$. In this work, we always set $N = (\rho\tau)^{-1}$ to make the success probability to be about $1 - e^{-1} \approx 0.63$.

C.3 The Analysis of the LLR Strategy Given in Algorithm 8

This strategy is implemented in Algorithm 8. Let $\mathbb{D} = \{\delta_0, \delta_1, \ldots, \delta_{s-1}\} \subseteq \mathbb{F}_2^m$ be a set of s *nonzero* differences. For each δ_i $(0 \le i < s)$, there is a set $\mathbb{M}_i = \{\lambda_{i,0}, \lambda_{i,1}, \ldots, \lambda_{i,\ell_i - 1}\}$ of ℓ_i linearly-independent linear masks, such that each $(\delta_i, \lambda_{i,j})$ forms a DL distinguisher with correlation $\mathfrak{c}_{i,j}$. For $\delta_i \in \mathbb{D}$, $0 \le j < \ell_i$, let $w_{i,j} = \lambda_{i,j} \cdot (y \oplus O)$. Let $\mathbf{D}_0^i$ and $\mathbf{D}_1^i$ be the distributions

$$\mathbf{D}_0^i : (\mathcal{B}(0, \pi_0^i), \ldots, \mathcal{B}(0, \pi_{\ell_i - 1}^i))$$
$$\mathbf{D}_1^i : (\mathcal{B}(0, 0.5), \ldots, \mathcal{B}(0, 0.5))$$

where $\pi_j^i = \frac{1 + \mathfrak{c}_{i,j}}{2}$, $0 \le j < \ell_i$ and $\mathcal{B}(0, p)$ is a Bernoulli distribution with parameter p. Let g_0^i and g_1^i be the probabilities that

$$\boldsymbol{w}_i = (w_{i,0}, \ldots, w_{i,\ell_i - 1})$$

is the result of sampling from $\mathbf{D}_0^i$ (i.e., from the real distribution) or $\mathbf{D}_1^i$ (i.e., the random distribution). Then,

$$g_0^i = \prod_{j=0}^{\ell_i - 1} (\pi_j^i)^{w_{i,j}} (1 - \pi_j^i)^{1 - w_{i,j}} \quad \text{and} \quad g_1^i = \prod_{j=0}^{\ell_i - 1} 2^{-1}.$$

Like [BBC+22], we define the LLR statistics as $\ln(\frac{g_0^i}{g_1^i})$ which is equal to

$$\frac{1}{2}\sum_{j=0}^{\ell_i-1}\ln(1-\mathfrak{c}_{i,j}^2)+\frac{1}{2}\sum_{j=0}^{\ell_i-1}(-1)^{w_{i,j}}\ln(\frac{1+\mathfrak{c}_{i,j}}{1-\mathfrak{c}_{i,j}})+\ell_i\ln 2.$$

In Algorithm 8, an element in a translation is accepted only when the LLR statistic is at least γ_i. Let $\theta : \mathbb{F}_2^{\ell_i}\times\mathbb{R}\to\{0,1\}$ be a function define as

$$\theta(u,\gamma_i)=\begin{cases}1 & \frac{1}{2}\sum_{j=0}^{\ell_i-1}\ln(1-\mathfrak{c}_{i,j}^2)+\frac{1}{2}\sum_{j=0}^{\ell_i-1}(-1)^{u_j}\ln\left(\frac{1+\mathfrak{c}_{i,j}}{1-\mathfrak{c}_{i,j}}\right)+\ell_i\ln 2\geq\gamma_i\\ 0 & \text{Otherwise}\end{cases}$$

where $u=(u_0,\cdots,u_{\ell_i-1})\in\mathbb{F}_2^{\ell_i}$. Let p_i be probability that the LLR statistic is at least γ_i when $F(x\oplus\delta_i)=O$ and q_i be probability that the LLR statistic is at least γ_i when $F(x\oplus\delta_i)\neq O$. We have

$$p_i=\sum_{\substack{u\in\mathbb{F}_2^{\ell_i},\\ \theta(u,\gamma_i)=1}}\prod_{j=0}^{\ell_i-1}\left(\frac{1}{2}+\frac{(-1)^{u_i}\mathfrak{c}_{i,j}}{2}\right)\quad\text{and}\quad q_i=\sum_{\substack{u\in\mathbb{F}_2^{\ell_i},\\ \theta(u,\gamma_i)=1}}2^{-\ell_i}.\tag{13}$$

Algorithm 8: The LLR-based statistical test to implement `PreTest()`

Input: $y=F(x)$ for some $x\in\mathbb{F}_2^m$, the preimage O, $\delta_i\in\mathbb{D}$, linear masks $\mathbb{M}_i=\{\lambda_{i,0},\ldots,\lambda_{i,\ell_i-1}\}$ such that $(\delta_i,\lambda_{i,j})$ is a differential-linear approximation of F with correlation $\mathfrak{c}_{i,j}$, and the threshold γ_i

Output: 0 or 1

LLR $\leftarrow \ell_i\ln 2$
for $0\leq j<\ell_i$ **do**
  LLR $\leftarrow$ LLR $+\frac{1}{2}\ln(1-\mathfrak{c}_{i,j}^2)+\frac{1}{2}(-1)^{\lambda_{i,j}\cdot(y\oplus O)\oplus\zeta_{\mathfrak{c}_{i,j}}}\ln(\frac{1+\mathfrak{c}_{i,j}}{1-\mathfrak{c}_{i,j}})$
if LLR $<\gamma_i$ **then**
  return 1
return 0

Complexity Analysis. When `PreTest()` is instantiated with Algorithm 8, $\mathbb{S}_{x,\mathbb{D}}=\{x\oplus\delta_i:\delta_i\in\mathbb{D},\text{LLR}\geq\gamma_i\}$. Thus, on average we expect $|\mathbb{S}_{x,\mathbb{D}}|=\sum_{i=0}^{s-1}q_i$ for a random x. Consequently, the complexity of Algorithm 1 is about $N\left(1+\sum_{i=0}^{s-1}q_i\right)$ evaluations of F.

Success Probability. The probability q of hitting a preimage in one while-loop of Algorithm 1 with a random guess $x\in\mathbb{F}_2^m$ can be computed as

$$q\geq\Pr[F(x)=O]+\sum_{i=0}^{s-1}\Pr[F(x\oplus\delta_i)=O\text{ and }x\oplus\delta_i\in\mathbb{S}_{x,\mathbb{D}}].\tag{14}$$

For $0 \le i < s$, we have

$$\begin{aligned}
& \Pr[F(x \oplus \delta_i) = O \text{ and } x \oplus \delta_i \in \mathbb{S}_{x,\mathbb{D}}] \\
= \ & \Pr[x \oplus \delta_i \in \mathbb{S}_{x,\mathbb{D}} \mid F(x \oplus \delta_i) = O] \ \Pr[F(x \oplus \delta_i) = O] \\
= \ & \Pr[x \oplus \delta_i \in \mathbb{S}_{x,\mathbb{D}} \mid F(x \oplus \delta_i) = O] \left(1 - \frac{1}{2^n}\right)^{i+1} \frac{1}{2^n} \\
= \ & p_i \left(1 - \frac{1}{2^n}\right)^{i+1} \frac{1}{2^n} > p_i \left(1 - \frac{1}{2^n}\right)^{s} \frac{1}{2^n},
\end{aligned} \tag{15}$$

where

$$p_i = \sum_{\substack{u \in \mathbb{F}_2^{\ell_i}, \\ \theta(u, \gamma_i) = 1}} \prod_{j=0}^{\ell_i - 1} \left(\frac{1}{2} + \frac{(-1)^{u_i} \mathfrak{c}_{i,j}}{2} \right).$$

Substituting Equation (15) into Equation (14) gives

$$q > \frac{1}{2^n} + \sum_{i=0}^{s-1} p_i \left(1 - \frac{1}{2^n}\right)^{s} \frac{1}{2^n}.$$

Since $s \ll 2^n$ and $\left(1 - \frac{1}{2^n}\right)^s = \left(1 - \frac{1}{2^n}\right)^{2^n \frac{s}{2^n}} \approx e^{-\frac{s}{2^n}} \approx 1$, we have

$$q > \frac{1}{2^n} + \sum_{i=0}^{s-1} \frac{p_i}{2^n} = 2^{\log(s+1)-n} \frac{1}{s+1} \left(1 + \sum_{i=0}^{s-1} p_i \right) = \rho\tau,$$

where $\tau = 2^{\log(s+1)-n}$ and $\rho = \frac{1}{s+1}(1 + \sum_{i=0}^{s-1} p_i)$. Therefore, the success probability that a preimage is detected after N while-loops of Algorithm 1 is lower bounded by $\mathsf{P}_{suc} = 1 - (1 - \rho\tau)^N$. In this work, we always set $N = (\rho\tau)^{-1}$ to make the success probability to be about $1 - e^{-1} \approx 0.63$.

D Success Probability of Algorithm 5

Since the translations $k \oplus \hat{\mathbb{D}}_K$ of $\hat{\mathbb{D}}_K$ with $k \in \hat{\mathbb{D}}_K^{\dashv}$ form a partition of $\mathbb{F}_2^m$, the correct key K must be in one of the translations for some k, where K is randomly chosen from $\mathbb{F}_2^m$. The probability q of hitting the correct key by Algorithm 5 can be estimated as

$$q = \Pr[k \oplus K = 0] + \sum_{i=0}^{s-1} \Pr[k \oplus K = \delta_i \ \texttt{and} \ k \oplus \delta_i \in \mathbb{S}_{k,\mathbb{D}_K}]. \tag{16}$$

For $0 \le i < s$, $\Pr[k \oplus K = \delta_i \text{ and } k \oplus \delta_i \in \mathbb{S}_{k,\mathbb{D}_K}]$ equals to

$$\Pr[k \oplus \delta_i \in \mathbb{S}_{k,\mathbb{D}_K} \mid k \oplus K = \delta_i] \ \Pr[k \oplus K = \delta_i] = p_i \frac{1}{s+1}, \tag{17}$$

where $p_i = \prod_{j=0}^{\ell_i-1}\left(\frac{1}{2}+\frac{|\mathfrak{c}_{i,j}|}{2}\right)$. Substituting Equation (17) into Equation (16) gives

$$q = \frac{1}{s+1} + \sum_{i=0}^{s-1}\frac{p_i}{s+1} = \frac{1}{s+1}\left(1+\sum_{i=0}^{s-1}p_i\right). \tag{18}$$

If we only use *deterministic* DL distinguishers as we do in all of our concrete cryptanalysis in this paper, the success probability of Algorithm 5 is about 1.

E Weaken the Conditions Imposed on the Differences

If the $\hat{\mathbb{D}}_K = \{0, \delta_0, \delta_1, \ldots, \delta_{s-1}\}$ is not a linear subspace, the key search can be accelerated with Algorithm 9. Since the translations $k \oplus \langle\hat{\mathbb{D}}_K\rangle$ of $\langle\hat{\mathbb{D}}_K\rangle$ with

---

Algorithm 9: Speed up the key-recovery with DL distinguishers

---

Input: $\mathbb{D} = \{(\delta_0,\delta_0'),\cdots,(\delta_{s-1},\delta_{s-1}')\} \subseteq \mathbb{F}_2^{m+n}$, and $\mathbb{M}_i = \{\lambda_{i,0},\ldots,\lambda_{i,\ell_i-1}\}$ for $0 \le i < s$ such that $((\delta_i,\delta_i'),\lambda_{i,j})$ is a *related-key* DL appoximation of F with correlation $\mathfrak{c}_{i,j}$, and $\hat{\mathbb{D}}_K = \{0\} \cup \{\delta_0,\cdots,\delta_{s-1}\}$.

Output: The master key K

Randomly choose a plaintext P, derive $C = F(K,P)$
for $0 \le i < s$ **do**
  $C_i = F(K, P \oplus \delta_i')$
for $k \in \langle\hat{\mathbb{D}}_K\rangle^{\dashv}$ **do**
  $k' \leftarrow$ A random element in $\langle\hat{\mathbb{D}}_K\rangle$
  $c \leftarrow F(k \oplus k', P)$
  if $c = C$ **then**
    if $F(k \oplus k', P \oplus \delta_i') = C_i, 0 \le i < s$ **then**
      return $k \oplus k'$ ▷ a few of $(P \oplus \delta_i', C_i)$ suffice
  for $0 \le i < s$ **do**
    $reject \leftarrow$ KeyTest$(c, C_i, (\delta_i,\delta_i'), \mathbb{M}_i)$
    if $reject = 0$ **then**
      if $F(k \oplus k' \oplus \delta_i, P \oplus \delta_i') = C_i, 1 \le i < s$ **then**
        return $k \oplus k'$ ▷ a few of $(P \oplus \delta_i', C_i)$ suffice

---

$k \in \langle\hat{\mathbb{D}}_K\rangle^{\dashv}$ form a partition of $\mathbb{F}_2^m$, the correct key K must be in one of the translations for some k, where K is randomly chosen from $\mathbb{F}_2^m$. The probability q of finding the correct key by Algorithm 5 can be computed as

$$q = \Pr[k \oplus k' = K] + \sum_{i=0}^{s-1}\Pr[k \oplus k' \oplus \delta_i = K \text{ and } k \oplus k' \oplus \delta_i \in \mathbb{S}_{k\oplus k',\mathbb{D}_K}]. \tag{19}$$

For $0 \le i < s$, we have

$$\begin{aligned}
& \Pr[k \oplus k' \oplus K = \delta_i \text{ and } k \oplus k' \oplus \delta_i \in \mathbb{S}_{k \oplus k', \mathbb{D}_K}] \\
= \ & \Pr[k \oplus k' \oplus \delta_i \in \mathbb{S}_{k \oplus k', \mathbb{D}_K} \mid k \oplus k' \oplus K = \delta_i] \ \Pr[k \oplus k' \oplus K = \delta_i] \\
= \ & \Pr[k \oplus k' \oplus \delta_i \in \mathbb{S}_{k \oplus k', \mathbb{D}_K} \mid k \oplus k' \oplus K = \delta_i] \ \frac{1}{|\langle \hat{\mathbb{D}}_K \rangle|} \\
= \ & p_i \frac{1}{|\langle \hat{\mathbb{D}}_K \rangle|},
\end{aligned} \tag{20}$$

where $p_i = \prod_{j=0}^{\ell_i - 1} \left(\frac{1}{2} + \frac{|\mathfrak{c}_{i,j}|}{2} \right)$. Substituting Equation (20) into Equation (19) gives

$$q = \frac{1}{|\langle \hat{\mathbb{D}}_K \rangle|} + \sum_{i=0}^{s-1} \frac{p_i}{|\langle \hat{\mathbb{D}}_K \rangle|} = \frac{1}{|\langle \hat{\mathbb{D}}_K \rangle|} \left(1 + \sum_{i=0}^{s-1} p_i \right) = \frac{s+1}{|\langle \hat{\mathbb{D}}_K \rangle|} \rho,$$

where $\rho = \frac{1}{s+1}(1 + \sum_{i=0}^{s-1} p_i)$. The complexity of Algorithm 9 is about

$$|\langle \hat{\mathbb{D}}_K \rangle^{\dashv}| \left(1 + \sum_{i=0}^{s-1} 2^{-\ell_i} \right)$$

evaluations of F. If we repeat Algorithm 9 $q^{-1} = \rho^{-1} \frac{|\langle \hat{\mathbb{D}}_K \rangle|}{s+1}$ times, the success probability is at least $1 - (1-q)^{q^{-1}} \approx 0.63$. The time complexity is about

$$\rho^{-1} \frac{|\langle \hat{\mathbb{D}}_K \rangle|}{s+1} |\langle \hat{\mathbb{D}}_K \rangle^{\dashv}| \left(1 + \sum_{i=0}^{s-1} 2^{-\ell_i} \right) = 2^{m - \log(s+1)} \rho^{-1} \left(1 + \sum_{i=0}^{s-1} 2^{-\ell_i} \right)$$

evaluations of F. Generally, the complexity of the inner products is negligible compared with the complexity due to the evaluations of F.

F Preimage Attacks on `XOEsch256`

In this section, we give the preimage attacks on `XOEsch`256 with an analogous method for `XOEsch`384. The notations are also similar to those used in the attacks on `XOEsch`384.

F.1 Preimage Attack on the 1.5-Step XOEsch-256

Our preimage attack works for 1.5-step `XOEsch`256 with a digest length of 128 bits, and to ensure the disjointness of the generated translations, it requires 2 message blocks (M_0, M_1). As shown in Figure 9, the 128-bit digest $(T_0, T_1) \in \mathbb{F}_2^{64 \times 2}$ can be inverted through `Alzette` ARX boxes A_{c_0} and A_{c_1}. Thus, if the linear masks employed for $(X_2^1, \ldots, X_5^1)$ in the attack are inactive, we can safely skip the `Alzette` ARX boxes in the last step. In addition, for any given M_0,

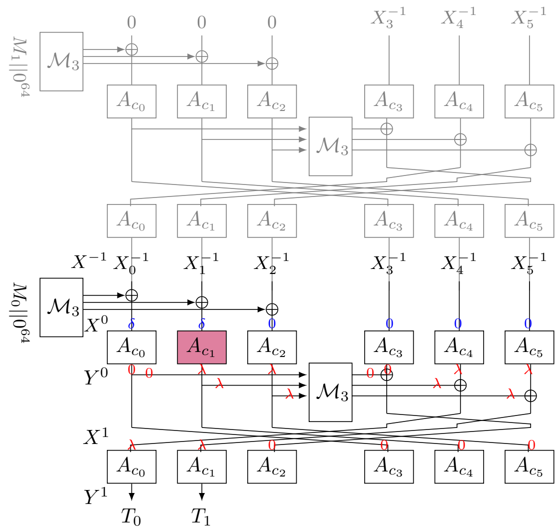


Fig. 9: Preimage attack on the 1.5-step XOEsch256.

X^{-1} can be derived. Consequently, we only need to focus on the function F_{LSM} : $\mathbb{F}_2^{128} \to \mathbb{F}_2^{128}$ mapping M_1 to (X_0^1, X_1^1).

The DL approximations for F_{LSM} are derived from DL distinguishers of Alzette. Given any DL approximation (δ, λ) of A_{c_1} with correlation $\mathfrak{c}$ listed in Table 4, we set the linear mask of X^1 to be $\Lambda(X^1) = (\lambda, \lambda, 0, 0, 0, 0)$. According to Lemma 4, the linear mask $\Lambda(Y^0)$ of Y^0 is $(0, \lambda, \lambda, 0, \lambda, \lambda)$. Let the difference of M_1 be $\Delta(M_1) = (\delta, \delta)$. According to Lemma 3, the difference of X^0 is $\Delta(X^0) = (\delta, \delta, 0, 0, 0, 0)$. As highlighted in Figure 9, only A_{c_1} has nonzero input difference and nonzero output linear mask at the same time. Therefore, the correlation of the above DL approximation for F_{LSM} is $\mathfrak{c}$.

The attack applies Algorithm 1 to F_{LSM} and proceeds as follows in the t-th while-loop of Algorithm 1. Set M_0 to be the 128-bit encoding of the integer t, and generate one random message block $M_1 \in \mathbb{F}_2^{128}$. Compute the value $\boldsymbol{x} = (x_0, x_1)$ for (X_0^1, X_1^1) from M_0 and M_1. If $\boldsymbol{x} = (x_0, x_1) = (A_{c_0}^{-1}(T_0), A_{c_1}^{-1}(T_1)) = (X_0^1, X_1^1)$, we are done with (M_0, M_1) being the preimage of (T_0, T_1). Otherwise, for each $\delta_i \in \mathbb{D}_{\texttt{Alzette}}$, we test whether $\lambda_{i,j} \cdot (\boldsymbol{x} \oplus (X_0^1, X_1^1)) = \zeta_{\mathfrak{c}_{i,j}}$ for all $\lambda_{i,j} \in \mathbb{M}_i$ ($\mathbb{D}_{\texttt{Alzette}}$ and $\mathbb{M}_i$ are given in Table 4). If δ_i passes the test, we compute the value $\boldsymbol{x}' = (x_0', x_1')$ for (X_0^1, X_1^1) from the message $(M_0, M_1 \oplus (\delta_i, \delta_i))$. If $\boldsymbol{x}' = (x_0', x_1') = (A_{c_0}^{-1}(T_0), A_{c_1}^{-1}(T_1))$, $(M_0, M_1 \oplus (\delta_i, \delta_i))$ is a preimage for (T_0, T_1). Note that with our approach for selecting (M_0, M_1), the translations we checked in the first N while-loops with $N < 2^{128}$ are guaranteed to be disjoint since the first 128 bits of two messages in the translations checked in different while-loops encode different integers.

Complexity and Success Probability. The digest length of this application is also $n = 128$. According to Table 4, the size of the set $\mathbb{D}$ of input differences is $s = |\mathbb{D}| = |\mathbb{D}_{\texttt{Alzette}}| = 15$, so $\rho \approx 2^{-0.26}$ and $\tau = 2^{\log(s+1)-n} = 2^{-124}$. The expectation of $|\mathbb{S}_{x,\mathbb{D}}| = \sum_{i=0}^{s-1} 2^{-\ell_i}$ is about $2^{-1.71}$. Thus, we set the number of translations checked to be $N = (\rho\tau)^{-1} = 2^{124.26}$ to make the success probability be 0.63. The time complexity of the attack can be estimated as $N(1+2^{-1.71}) = 2^{124.26} \times (1+2^{-1.71}) \approx 2^{124.64}$ evaluations of F_{LSM}, where $N = 2^{124.26}$ is the number of translations checked in the attack.

The needed $N = 2^{124.26}$ randomly-guessed translations required by Algorithm 1 can be selected by randomly choosing, e.g., $2^{100.26}$ M_0 and under each chosen M_0 we choose 2^{24} M_1 randomly. With such a skill, the computation of M_0 is negligible compared to the other parts. The core process of our attack is to apply Algorithm 1 to F_{LSM}. Considering that the nonlinear operations in `XOEsch` is much more costly than the linear layer, we approximately regard the cost of F_{LSM} as that of one step of `Sparkle`192. The 1.5-step `XOEsch`256 instance with a 128-bit digest at best requires one 1.5-step `Sparkle`192 (2 nonlinear layers), so the complexity of our attack is approximately $2^{123.64}$ 1.5-step `XOEsch`256 conductions.

F.2 Preimage Attack on the 2.5-Step XOEsch256

Our second application is to the 2.5-step `XOEsch`256. Akin to the preimage attack on the 2.5-step `XOEsch`384, we take the 128-bit-digest instance of `XOEsch`256 as an example. To ensure the disjointness of the generated cosets, this attack requires 2^{127} cosets of a 1-dimensional linear space, so we use 2 message blocks denoted by (M_0, M_1) (see Figure 10).

The 128-bit digest $(T_0, T_1) \in \mathbb{F}_2^{64\times 2}$ can be inverted through `Alzette` ARX boxes A_{c_0} and A_{c_1}. Thus, if the linear masks employed for $(X_2^2, \ldots, X_5^2)$ in the attack are inactive, we can safely skip the `Alzette` ARX boxes in the last step. In addition, when we choose an M_0, X^{-1} will be obtained. Consequently, in our preimage attack on the 2.5-step `XOEsch`256, we only need to focus on the second message block, i.e., M_1. Different from the 1.5-step attack, the function that we apply Algorithm 1 to is $F_{LSL} : \mathbb{F}_2^{192} \to \mathbb{F}_2^{128}$ that maps (Y_0^0, Y_1^0, Y_2^0) to (X_0^2, X_1^2), rather than the mapping that sends M_1 to (X_0^2, X_1^2), because the 2.5-step `Sparkle`384 is more complicated and more difficult to allow DL distinguishers.

Next, we introduce the DL distinguishers for F_{LSL}. Given any DL approximation (δ, λ) of A_{c_i} with correlation ϵ, we set the linear mask of X^2 to be $\Lambda(X^2) = (\lambda, \lambda, 0, 0, 0, 0)$. According to Lemma 4, the linear mask $\Lambda(Y^1)$ of Y^1 is $(0, \lambda, \lambda, 0, \lambda, \lambda)$. For the difference of Y^0, we set it to be $\Delta(Y^0) = (\delta, \delta, 0, 0, 0, 0)$. The difference of X^1 will be $\Delta(X^1) = (\delta, \delta, 0, \delta, \delta, 0)$, according to Lemma 3. Now, as highlighted in Figure 10, only the input difference and output linear mask of A_{c_1} and A_{c_4} in the second step are both nonzero. Therefore, the correlation of the above DL approximation for F_{LSL} is ϵ^2.

When applying Algorithm 1 to F_{LSL}, under each M_0 that we have chosen, we need to guess and check a value for (Y_0^0, Y_1^0, Y_2^0), say $\boldsymbol{y} = (y_0, y_1, y_2)$, and quickly check $\boldsymbol{y'} = (y_0 \oplus \delta, y_1 \oplus \delta, y_2)$ with the DL distinguishers. In this process,

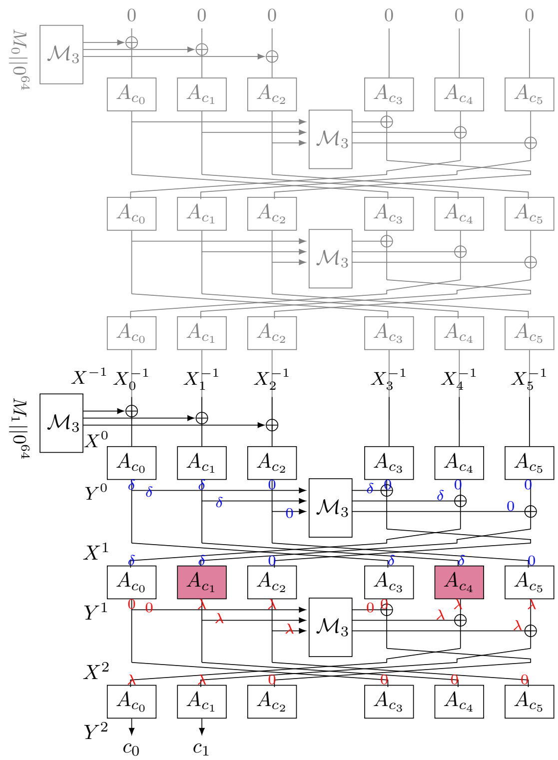


Fig. 10: Illustration of the preimage attacks on the 2.5-step XOEsch256.

both $\boldsymbol{y}$ and $\boldsymbol{y}'$ are possible to be a preimage of (T_0, T_1). However, due to the existence of $\mathcal{M}_3$ in the absorption phase and more critically, the second 128-bit input of this $\mathcal{M}_3$ should be 0 (see Figure 1), there is a risk that the recovered $\boldsymbol{y}$ or $\boldsymbol{y}'$ does not correspond to any valid M_1.

To address this risk, we can reuse the pre-computated $\mathbb{S}_\delta$ in Section 5.3. When X^{-1} is known, based on any $(\gamma_0, \gamma_1, \delta) \in \mathbb{S}_\delta$ we choose $\boldsymbol{y}$ and $\boldsymbol{y}'$ such that both $\boldsymbol{y}$ and $\boldsymbol{y}'$ can lead to a valid M_1 in the following way,

$$\begin{cases} \boldsymbol{y} = (y_0, y_1, y_2) = (\gamma_0, \gamma_1, \gamma_2) \\ \boldsymbol{y}' = (y_0 \oplus \delta, y_1 \oplus \delta, y_2) = (\gamma_0 \oplus \delta, \gamma_1 \oplus \delta, \gamma_2) \end{cases} \tag{21}$$

where

$$\begin{cases} (u_j, v_j) = A_{c_j}^{-1}(\gamma_j) \oplus X_j^{-1}, j \in \{0, 1\} \\ \gamma_2 = A_{c_2}\left((\ell(v_0, v_1), \ell(u_0, u_1)) \oplus X_2^{-1}\right) \end{cases}.$$

It can be checked that $\boldsymbol{y} = (y_0, y_1, y_2)$ and $\boldsymbol{y}' = (y_0 \oplus \delta, y_1 \oplus \delta, y_2)$ respectively guarantee that

$$\left(A_{c_0}^{-1}(y_0), A_{c_1}^{-1}(y_1), A_{c_2}^{-1}(y_2)\right) \oplus \left(X_0^{-1}, X_1^{-1}, X_2^{-1}\right)$$

and

$$(A_{c_0}^{-1}(y_0 \oplus \delta), A_{c_1}^{-1}(y_1 \oplus \delta), A_{c_2}^{-1}(y_2)) \oplus (X_0^{-1}, X_1^{-1}, X_2^{-1})$$

satisfy Lemma 5 (it this XOEsch256 case, the w in Lemma 5 should be instanced as 3). Hence, no matter whether Algorithm 1 returns $\boldsymbol{y}$ from Line 7 or $\boldsymbol{y}'$ from Line 13, we are sure that M_1 exists.

This attack also uses the group of DL distinguishers given in Equation 7. It proceeds as follows in each while-loop of Algorithm 1. Set M_0 to be the 128-bit encoding of the integer t. The corresponding X^{-1} can be derived. Under each X^{-1}, we choose one $(\gamma_0, \gamma_1, \delta)$ in $\mathbb{S}_\delta$ and generate $\boldsymbol{y}$ according to Equation (21). Compute the value $\boldsymbol{x} = (x_0, x_1)$ for (X_0^1, X_1^1) from M_0 and $\boldsymbol{y}$. If $\boldsymbol{x} = (x_0, x_1) = (A_{c_0}^{-1}(T_0), A_{c_1}^{-1}(T_1)) = (X_0^2, X_1^2)$, we are done with $(M_0, \boldsymbol{y})$ that can lead to a preimage of (T_0, T_1) according to Equation (4). Otherwise, for $\boldsymbol{y}' = \boldsymbol{y} \oplus (\delta, \delta, 0, 0)$, we test whether $\lambda \cdot (\boldsymbol{x} \oplus (X_0^2, X_1^2)) = \zeta_{\epsilon_j}$ for all $\lambda_j \in \mathbb{M}$. If $\boldsymbol{y}'$ passes the test, we compute the value $\boldsymbol{x}' = (x_0', x_1')$ for (X_0^2, X_1^2) from M_0 and $\boldsymbol{y}'$. If $\boldsymbol{x}' = (x_0', x_1') = (A_{c_0}^{-1}(T_0), A_{c_1}^{-1}(T_1)) = (X_0^2, X_1^2)$, we can compute the preimage for (T_0, T_1) from $(M_0, \boldsymbol{y}')$ following Equation (4).

Complexity and Success Probability. The output of F_{LSL} is $n = 128$. Since we only use one difference, the size of the set $\mathbb{D}$ of input differences is $s = |\mathbb{D}| = 1$, so $\rho \approx 2^{-0.01}$ and $\tau = 2^{\log(s+1)-n} = 2^{-127}$. The expectation of $|\mathbb{S}_{x,\mathbb{D}}| = \sum_{i=0}^{s-1} 2^{-\ell_i}$ is about 2^{-8}. Thus, to make the success probability of this attack be about 0.63, we set $N = (\rho\tau)^{-1} = 2^{127.01}$. The time complexity of the attack can be estimated as $N(1 + 2^{-8}) = 2^{127.01} \times (1 + 2^{-8}) \approx 2^{127.02}$ evaluations of F_{LSL}.

In our attack, the selection of the $N = 2^{127.02}$ cosets can be optimized by randomly choosing, e.g., $2^{117.02}$ M_0 and under each chosen M_0 we traverse all 2^{10} $(\gamma_0, \gamma_1, \delta)$ in $\mathbb{S}_\delta$. With this technique, the computation of M_0 is negligible compared to other parts. Generating $\boldsymbol{y}$ from $(\gamma_0, \gamma_1, \delta)$ costs 3 Alzette operations. Further, when pre-computing $(\gamma_0, \gamma_1, \delta)$, we can actually store $(A_{c_0}^{-1}(\gamma_0), A_{c_1}^{-1}(\gamma_1))$. Thus, the cost can be reduced to 1 Alzette operations (1/6 steps of Sparkle384). Moreover, considering that the nonlinear operations in XOEsch is much more costly than the linear layer, we approximately regard the cost of F_{LSL} as that of one step of Sparkle384. Thus, to check $\boldsymbol{y}$ costs us about $1 + 1/6$ steps of Sparkle384. The 2.5-step XOEsch384 instance with a 128-bit digest requires about one 2.5-step Sparkle384 (3 nonlinear layers). Consequently, the complexity of the attack is approximately $2^{127.02} \times 7/18 \approx 2^{125.66}$ 2.5-step XOEsch256 evaluations.

G Preimage Attack on the 2.5-Step Variant XOEsch

In this section, we give preimage attacks on variants of the 2.5-step XOEsch384 and XOEsch256 where the first two Alzette's in the first step are parameterized by the same constant. If the two Alzette's use the same parameters, our attack can have a better complexity, which provides justification for the designers' choice to parameterize different Alzette with different constants. For convenience, we denote the variants of XOEsch384 and XOEsch256 by XOEsch⋆384 and XOEsch⋆256, respectively. Their underlying permutations are also variant of Sparkle, denoted by Sparkle⋆512 and Sparkle⋆384, respectively.

G.1 Preimage Attack on the 2.5-Step Variant XOEsch384

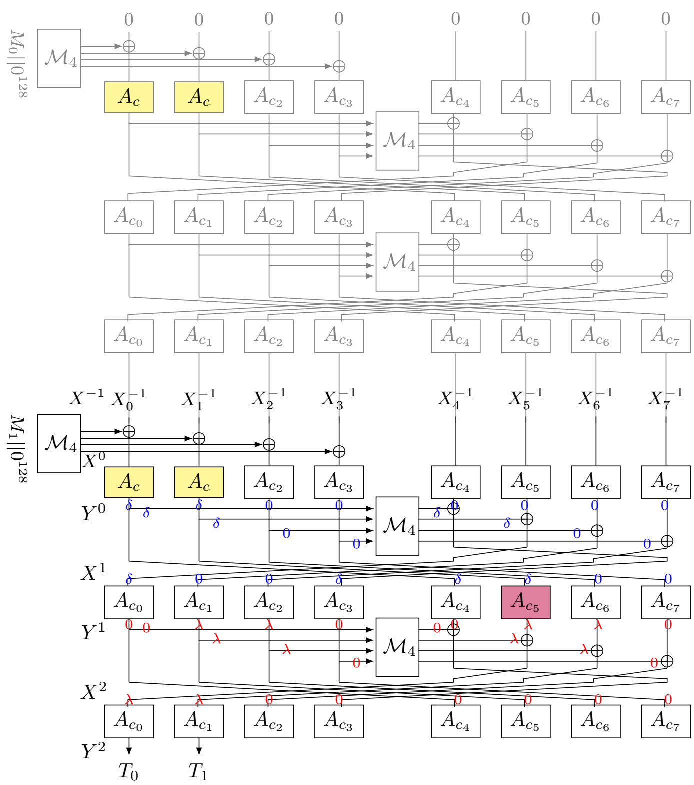


Fig. 11: Illustration of the preimage attack on the 2.5-step variant XOEsch384, note that the first two Alzette's in the first step are parameterized by the same constant.

The preimage attack on the 2.5-step XOEsch⋆384 is similar to that on the 2.5-step XOEsch384, except that for any (y_0, y_1) satisfying $y_0 = y_1$, it is free to obtain

$$A_c^{-1}(y_0) \oplus A_c^{-1}(y_0 \oplus \delta) = A_c^{-1}(y_1) \oplus A_c^{-1}(y_1 \oplus \delta).$$

Hence, in the attack, we can choose (y, y) for (Y_0^0, Y_1^0), and calculate (y_2, y_3) for Y_2^0, Y_3^0 to satisfy the Lemma 5 to guarantee that there exists an M_1. This means all DL distinguishers in Table 4 can be used in this attack. The attack process is almost the same with Section 5.3, so we omit the details.

Complexity and Success Probability. The digest length is still $n = 128$. The difference set has a size $s = \mathbb{D}_{\texttt{Alzette}} = 15$, so $\rho \approx 2^{-0.36}$ and $\tau = 2^{\log(s+1)-n} = 2^{124}$. The expectation of $|\mathbb{S}_{x,\mathbb{D}}|$ is about $2^{-1.67}$.

To make the success probability be 0.63, we set $N = (\rho\tau)^{-1} = 2^{124.36}$. The time complexity is then $N(1 + |\mathbb{S}_{x,\mathbb{D}}|) = 2^{124.36} \times (1 + 2^{-1.67}) \approx 2^{124.75}$ F_{LSL} operations.

Considering that the nonlinear operations in `XOEsch`* is much more costly than the linear layer, by counting the number of involved `Alzette` we can regard the cost of the 1.5 step of `Sparkle`*512 as 1/2 of the 2.5-step `Sparkle`*512. Thus, the time complexity is $2^{123.75}$ 2.5-step `Sparkle`*512 operations. Considering that `XOEsch`*384 with a 128-bit operation only generates one block of digest, so at the best case one execution of `XOEsch`*384 costs only one 2.5-step `Sparkle`*512. Similar to the preimage attack on the 2.5-step `XOEsch`384, on average one guess of our attack costs approximately only one 2.5-step `Sparkle`*512. As a result, our complexity is still $2^{123.75}$ 2.5-step `XOEsch`*384.

Complexity for the `XOEsch`*384 with a 192-Bit Digest. In the case of a 192-bit output, the digest consists of 2 blocks. A similar attack as the above one can be mounted, where the messages are also 2-block ones (that can be split into two phases to choose). The two digest blocks have little influence on our attack, except that when our guess matches the first block, we need to continue to match the second block. Since the probability that the first block is matched is very small, the cost for the second matching is negligible. The final complexity is about $2^{186.75}$ 2.5-step `XOEsch`*384 calculations. The successful probability is still about 0.63.

G.2 Preimage Attack on the 2.5-Step Variant `XOEsch256`

The preimage attack on the 2.5-step `XOEsch`*256 is also similar to that on the `XOEsch`256, except that for any (y_0, y_1) satisfying $y_0 = y_1$, it is free to obtain

$$A_c^{-1}(y_0) \oplus A_c^{-1}(y_0 \oplus \delta) = A_c^{-1}(y_1) \oplus A_c^{-1}(y_1 \oplus \delta).$$

Hence, we only need to choose (y, y) for (Y_0^0, Y_1^0), and compute y_2 for Y_2^0 to satisfy the Lemma 5 to make sure that there must exist an M_1. This means all DL distinguishers in Table 4 can be used in this attack. Thus, we will not use the DL distinguishers in Equation (7). Instead, we will use all the 16 groups of DL distinguishers in Table 4. See Figure 12, every DL distinguisher for `Alzette` with a correlation of $\mathfrak{c}$ is mapped to one DL distinguisher for `XOEsch`*256 with a correlation of $\mathfrak{c}^2$.

Complexity and Success Probability. The digest length is $n = 128$. According to Table 4, the difference set has a size $s = |\mathbb{D}_{\texttt{Alzette}}| = 15$, so $\rho \approx 2^{-0.67}$ and $\tau = 2^{\log(s+1)-n} = 2^{-124}$. The expectation of $|\mathbb{S}_{x,\mathbb{D}}|$ is about $2^{-1.67}$. To make the success probability be 0.63, we set $N = (\rho\tau)^{-1} = 2^{124.67}$. The time complexity is then $N(1 + |\mathbb{S}_{x,\mathbb{D}}|) = 2^{124.67} \times (1 + 2^{-1.67}) \approx 2^{125.06}$.

In the above attack process, for every guess of M_0 and (X_0^0, X_1^0), we only need to compute (X_0^2, X_1^2), which costs 1 step of `Sparkle`*384. Considering that

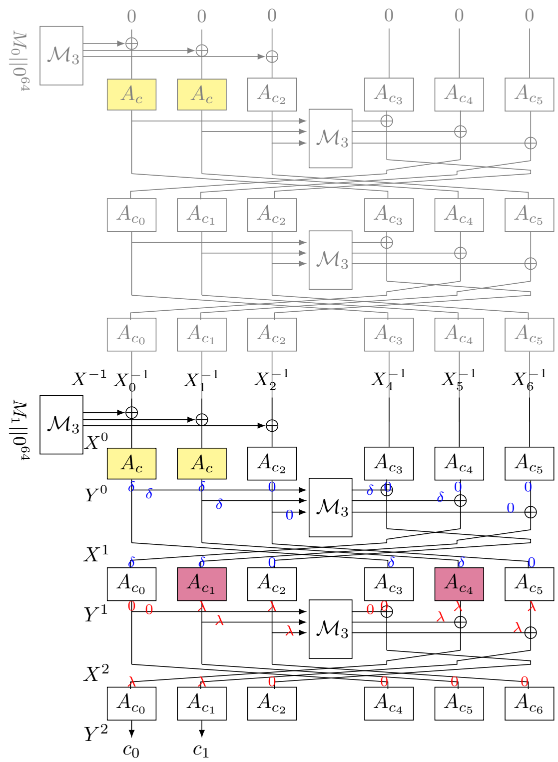


Fig. 12: Illustration of the preimage attacks on the 2.5-step XOEsch*256.

the nonlinear operations in XOEsch* is much more costly than the linear layer, by counting the number of involved Alzette we can regard the cost of the 1.5 step of Sparkle*384 as 1/2 the 2.5-step Sparkle*384. Thus, the time complexity is $2^{124.06}$ 2.5-step Sparkle384 operations.

Considering that XOEsch*256 with a 128-bit operation only generates one block of digest, so at the best case one execution of XOEsch*256 costs only one 1.5-step Sparkle*384. Though our attack uses 2 block messages, we can actually randomly select many M_0 and under each of them we randomly select (X_0^0, X_1^0) to construct sufficient translations of $\hat{\mathbb{D}}_{\mathsf{Alzette}}$. Obviously, we only need to calculate M_0 once for all its corresponding (X_0^0, X_1^0). Thus, on average one guess of our attack also costs approximately only one 2.5-step Sparkle*384. As a result, our complexity is still $2^{124.06}$ 2.5-step XOEsch*256.

H Specification of the Ascon Hash Family

The hash functions in the Ascon family adopt the sponge mode [BDPA08] as illustrated in Figure 13. Both the hash functions with fixed output size (Ascon-Hash and Ascon-Hasha) and the XOFs with variable output size (Ascon-XOF

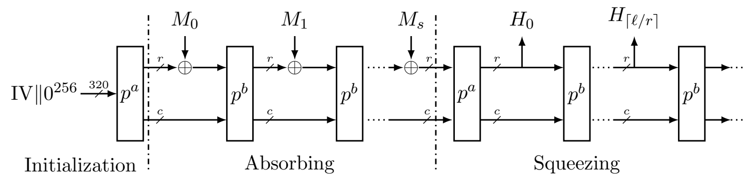


Fig. 13: The hash function structure of `Ascon` hash family

and `Ascon-XOFa`) internally use the same permutation with different rounds. The

Table 5: Parameters for `Ascon-XOF` and `Ascon-XOFa`.

Target	Size of					Rounds		IV
	State	Rate	Capacity	Digest	Pre. Sec.	p^a	p^b	
`Ascon-XOF`	320	64	256	ℓ	$\min(128, \ell)$	12	12	`00400c0000000000`
`Ascon-XOFa`	320	64	256	ℓ	$\min(128, \ell)$	12	8	`00400c0400000000`

structure of `Ascon` hash family is shown in Figure 13. p^a and p^b are iterative permutations with a and b rounds, respectively. Since this paper focuses on the `Ascon-XOF`, we list the parameters used for `Ascon-XOF` and `Ascon-XOFa` in Table 5. This paper targets 3- and 4-round `Ascon-XOF`, which is naturally applicable to `Ascon-XOFa`.

The round function $p = p_L \circ p_S \circ p_C$ operates on a 320-bit state arranged into five 64-bit words. The three components are described as follows,

Addition of Constants (p_C). An 8-bit constant is XORed to the bit positions $56, \ldots, 63$ of the second 64-bit word at each round.

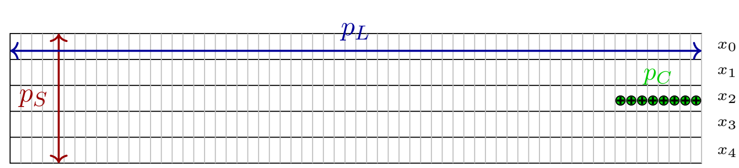


Fig. 14: The state of `Ascon` permutation, and illustration of p_C, p_S and p_L.

Substitution Layer (p_S). Update each slice of the 320-bit state by applying the 5-bit S-box defined by the following algebraic normal forms:

$$\begin{cases} y_0 = x_4x_1 + x_3 + x_2x_1 + x_2 + x_1x_0 + x_1 + x_0 \\ y_1 = x_4 + x_3x_2 + x_3x_1 + x_3 + x_2x_1 + x_2 + x_1 + x_0 \\ y_2 = x_4x_3 + x_4 + x_2 + x_1 + 1 \\ y_3 = x_4x_0 + x_4 + x_3x_0 + x_3 + x_2 + x_1 + x_0 \\ y_4 = x_4x_1 + x_4 + x_3 + x_1x_0 + x_1 \end{cases} \quad \begin{cases} y_0 \leftarrow \Sigma_0(x_0) = x_0 + (x_0 \ggg 19) + (x_0 \ggg 28) \\ y_1 \leftarrow \Sigma_1(x_1) = x_1 + (x_1 \ggg 61) + (x_1 \ggg 39) \\ y_2 \leftarrow \Sigma_2(x_2) = x_2 + (x_2 \ggg 1) + (x_2 \ggg 6) \\ y_3 \leftarrow \Sigma_3(x_3) = x_3 + (x_3 \ggg 10) + (x_3 \ggg 17) \\ y_4 \leftarrow \Sigma_4(x_4) = x_4 + (x_4 \ggg 7) + (x_4 \ggg 41) \end{cases}$$

Linear Diffusion Layer (p_L). Apply a linear transformation Σ_i to each 64-bit word y_i with $0 \le i < 5$, where Σ_i is defined as above.

I Improved Preimage Attack on 4-round `Ascon-XOF` with the Maximum Likelihood Strategy

The DL distinguishers employed in the 4-round attack are produced with $\mathbb{D} = \{\delta_0 = (0), \cdots, \delta_{62} = (62)\}$ and the corresponding

$$\mathbb{M}_i = \{(i+8), (i+30), (i+50), (i+54), (i+27), (i+47)\}, \quad 0 \le i < 63.$$

Note that according to the padding rule of the `Ascon-XOF`, the message is padded with at least one "1" bit, and thus the last bit of the difference of the messages cannot be active, which is reflected by $(63) \notin \mathbb{D}$. The absolute correlations of the 4-round distinguishers for all $0 \le i < 63$ are listed as follows:

$$(i) \xrightarrow[0.25]{4R} (i+8), (i) \xrightarrow[0.25]{4R} (i+30), (i) \xrightarrow[0.44]{4R} (i+50), (i) \xrightarrow[0.50]{4R} (i+54),$$

$$(i) \xrightarrow[0.14]{4R} (i+27), (i) \xrightarrow[0.16]{4R} (i+47).$$

Since $\hat{\mathbb{D}}$ is not a linear space, we have to choose the translations of $\hat{\mathbb{D}}$ in a sufficiently large space to guarantee the disjointness. For `Ascon-XOF` with a 128-bit digest, we need approximately $2^{128-\log(|\hat{\mathbb{D}}|)} = 2^{128-\log(64)} = 2^{122}$ translations. As shown in Figure 5, if we use 5-block messages $(M_0, M_1, M_2, M_3, M_4) \in \mathbb{F}_2^{64\times 5}$ to randomize the selection of the 2^{122} translations, then the probability that they are not disjoint is about $(64^2 \times 2^{244})/2^{321} \approx 2^{-65}$ according to Lemma 2, which is negligible.[5]

Given the 128-bit hash digest $(T_0, T_1) \in \mathbb{F}_2^{64\times 2}$ of `Ascon-XOF`, to recover the preimage $(M_0, M_1, M_2, M_3, M_4)$, we apply Algorithm 1 to the function mapping $(M_0, M_1, M_2, M_3, M_4)$ to (T_0, T_1), where the input differences of the distinguishers are injected through M_4 and the linear masks are applied to T_0. In the attack, we first randomly choose a value for (M_0, M_1, M_2, M_3) and generate the

[5] We can also choose the translations $x \oplus \hat{\mathbb{D}}$ by selecting x only in $\langle \hat{\mathbb{D}} \rangle^{\dashv}$, but this will increase the time complexity by a factor of 2. Because for each while-loop in Algorithm 1, two `Ascon` permutations are evaluated while the current method requires one.

intermediate state X right before the absorbing of M_4. Then, based on X and M_4 we compute the value $x_0 \in \mathbb{F}_2^{64}$ for T_0. If $x_0 = T_0$, we continue to generate x_1 and check if $x_1 = T_1$. If $(x_0, x_1) = (T_0, T_1)$, $(M_0, M_1, M_2, M_3, M_4)$ is then a preimage. Otherwise, for $\delta_i \in \mathbb{D}$, we check if $\lambda \cdot (x_0 \oplus T_0) = \zeta_{\mathfrak{c}_{i,j}}$ holds for all $0 \leq j < 4$. If δ_i passes the `PreTest()`, where `PreTest()` is instantiated with Algorithm 4, we use X and $M_4 \oplus \delta_i$ to generate x_0' and check if $x_0' = T_0$. If so, we continue to generate x_1' and check whether $x_1' = T_1$. If $(x_0', x_1') = (T_0, T_1)$, $(M_0, M_1, M_2, M_3, M_4 \oplus \delta_i)$ is a preimage of (T_0, T_1). In addition, for $0 \leq i < 64$, we set $\gamma_i = 0.0504984$ and

$$\mathcal{N}_{\gamma_i} = \{(0,0,0,0,0,0), (0,0,0,0,1,0), (0,0,0,0,0,1)\}.$$

We define $L^{(i)} : \mathbb{F}_2^{64\times 5} \mapsto \mathbb{F}_2^6$ to be the function mapping $x \in \mathbb{F}_2^{64\times 5}$ to

$$\begin{pmatrix} e_{i+8} \cdot (F(x) \oplus F(x \oplus \delta_i)), \\ e_{i+30} \cdot (F(x) \oplus F(x \oplus \delta_i)), \\ e_{i+50} \cdot (F(x) \oplus F(x \oplus \delta_i)), \\ e_{i+54} \cdot (F(x) \oplus F(x \oplus \delta_i)), \\ e_{i+27} \cdot (F(x) \oplus F(x \oplus \delta_i)), \\ e_{i+47} \cdot (F(x) \oplus F(x \oplus \delta_i)) \end{pmatrix}^T$$

Only when

$$\begin{pmatrix} e_{i+8} \cdot (F(x) \oplus O), \\ e_{i+30} \cdot (F(x) \oplus O), \\ e_{i+50} \cdot (F(x) \oplus O), \\ e_{i+54} \cdot (F(x) \oplus O), \\ e_{i+27} \cdot (F(x) \oplus O), \\ e_{i+47} \cdot (F(x) \oplus O) \end{pmatrix}^T \in \mathcal{N}_{\gamma_i},$$

Algorithm 4 will output 0. For $0 \leq j \leq 63$, $e_j \cdot (F(x) \oplus F(x \oplus \delta_i))$ is the output of different S-box in `Ascon` permutation, then we can consider these D-L approximation with same input difference is independent with each other. Thus, according to Equation (12), for $u \in \mathcal{N}_{\gamma_i}$, $g_u^{(i)} = \Pr_{x \in \mathbb{F}_2^{5\times 64}}[L^{(i)}(x) = u]$ is equal to

$$g_{(0,0,0,0,0,0)}^{(i)} = \left(\frac{1}{2} + \frac{0.25}{2}\right)^2 \left(\frac{1}{2} + \frac{0.44}{2}\right) \left(\frac{1}{2} + \frac{0.5}{2}\right) \left(\frac{1}{2} + \frac{0.14}{2}\right) \left(\frac{1}{2} + \frac{0.26}{2}\right) = 0.0697359$$

$$g_{(0,0,0,0,0,1)}^{(i)} = \left(\frac{1}{2} + \frac{0.25}{2}\right)^2 \left(\frac{1}{2} + \frac{0.44}{2}\right) \left(\frac{1}{2} + \frac{0.5}{2}\right) \left(\frac{1}{2} + \frac{0.14}{2}\right) \left(\frac{1}{2} - \frac{0.26}{2}\right) = 0.0526078$$

$$g_{(0,0,0,0,1,0)}^{(i)} = \left(\frac{1}{2} + \frac{0.25}{2}\right)^2 \left(\frac{1}{2} + \frac{0.44}{2}\right) \left(\frac{1}{2} + \frac{0.5}{2}\right) \left(\frac{1}{2} - \frac{0.14}{2}\right) \left(\frac{1}{2} + \frac{0.26}{2}\right) = 0.0504984$$

So we can get $q_i = \frac{3}{64}$ and

$$p_i = g_{(0,0,0,0,0,0)}^{(i)} + g_{(0,0,0,0,0,1)}^{(i)} + g_{(0,0,0,0,1,0)}^{(i)} = 0.1728421$$

Complexity and Success Probability. The output length in this application is $n = 128$. According to our DL distinguishers, the size of the set $\mathbb{D}$ of input differences is $s = |\mathbb{D}| = 63$, so $\rho = \frac{1}{64}(1+\sum_{i=0}^{62} p_i) \approx 2^{-2.43}$ and $\tau = 2^{\log(s+1)-n} = 2^{-122}$. The expectation of $|\mathbb{S}_{x,\mathbb{D}}| = \sum_{i=0}^{62} q_i$ is about $2^{1.56}$. Thus, we let $N = (\rho\tau)^{-1} = 2^{124.16}$ to make the success probability of this attack be 0.63. The time complexity of the attack can be estimated as $N(1+2^{1.98}) = 2^{124.43} \times (1+2^{1.56}) \approx 2^{126.41}$ evaluations of 4-round `Ascon` permutation. In our attack, the selection of the $N = 2^{124.43}$ translations can be optimized by randomly choosing, e.g., $2^{104.43}$ (M_0, M_1, M_2, M_3) and under each chosen (M_0, M_1, M_2, M_3) we choose 2^{20} M_4 randomly. With this technique, the computation of (M_0, M_1, M_2, M_3) is negligible compared to other parts. Considering that `Ascon-XOF` with a 128-bit digest requires at least 2 `Ascon` permutations. Our complexity can be scaled to $2^{125.41}$ 4-round `Ascon-XOF` operations. The memory cost is negligible. Compared with the strictest approach, there are $2^{0.06}$ improved.

J Improved Preimage Attack on 4-round `Ascon-XOF` with the LLR Strategy

The DL distinguishers employed in the 4-round attack are produced with $\mathbb{D} = \{\delta_0 = (0), \cdots, \delta_{62} = (62)\}$ and the corresponding

$$\mathbb{M}_i = \{(i+8), (i+30), (i+50), (i+54), (i+27), (i+47)\}, \quad 0 \le i < 63.$$

Note that according to the padding rule of the `Ascon-XOF`, the message is padded with at least one "1" bit, and thus the last bit of the difference of the messages cannot be active, which is reflected by $(63) \notin \mathbb{D}$. The absolute correlations of the 4-round distinguishers for all $0 \le i < 63$ are listed as follows:

$$(i) \xrightarrow[0.25]{4R} (i+8), (i) \xrightarrow[0.25]{4R} (i+30), (i) \xrightarrow[0.44]{4R} (i+50), (i) \xrightarrow[0.50]{4R} (i+54),$$

$$(i) \xrightarrow[0.14]{4R} (i+27), (i) \xrightarrow[0.16]{4R} (i+47).$$

Since $\hat{\mathbb{D}}$ is not a linear space, we have to choose the translations of $\hat{\mathbb{D}}$ in a sufficiently large space to guarantee the disjointness. For `Ascon-XOF` with a 128-bit digest, we need approximately $2^{128-\log(|\hat{\mathbb{D}}|)} = 2^{128-\log(64)} = 2^{122}$ translations. As shown in Figure 5, if we use 5-block messages $(M_0, M_1, M_2, M_3, M_4) \in \mathbb{F}_2^{64\times 5}$ to randomize the selection of the 2^{122} translations, then the probability that they are not disjoint is about $(64^2 \times 2^{244})/2^{321} \approx 2^{-65}$ according to Lemma 2, which is negligible.[6]

Given the 128-bit hash digest $(T_0, T_1) \in \mathbb{F}_2^{64\times 2}$ of `Ascon-XOF`, to recover the preimage $(M_0, M_1, M_2, M_3, M_4)$, we apply Algorithm 1 to the function mapping $(M_0, M_1, M_2, M_3, M_4)$ to (T_0, T_1), where the input differences of the distinguishers are injected through M_4 and the linear masks are applied to T_0. In the

[6] We can also choose these translations $x \oplus \hat{\mathbb{D}}$ by selecting x only in $\langle\hat{\mathbb{D}}\rangle^{\perp}$, but this will increase the time complexity by a factor of 2. Because for each while-loop in Algorithm 1, two `Ascon` permutations are evaluated while the current method requires one.

attack, we first randomly choose a value for (M_0, M_1, M_2, M_3) and generate the intermediate state X right before the absorbing of M_4. Then, based on X and M_4 we compute the value $x_0 \in \mathbb{F}_2^{64}$ for T_0. If $x_0 = T_0$, we continue to generate x_1 and check if $x_1 = T_1$. If $(x_0, x_1) = (T_0, T_1)$, $(M_0, M_1, M_2, M_3, M_4)$ is then a preimage. Otherwise, for $\delta_i \in \mathbb{D}$, we check if $\lambda \cdot (x_0 \oplus T_0) = \zeta_{\mathfrak{c}_{i,j}}$ holds for all $0 \leq j < 4$. If δ_i passes the `PreTest()`, where `PreTest()` is instantiated with Algorithm 8, we use X and $M_4 \oplus \delta_i$ to generate x'_0 and check if $x'_0 = T_0$. If so, we continue to generate x'_1 and check whether $x'_1 = T_1$. If $(x'_0, x'_1) = (T_0, T_1)$, $(M_0, M_1, M_2, M_3, M_4 \oplus \delta_i)$ is a preimage of (T_0, T_1). In addition, for $0 \leq i < 64$, we set $\gamma_i = 6.988$ and

$$\mathbb{M}_i = \{(i+8), (i+30), (i+50), (i+54), (i+27), (i+47)\}.$$

For $u \in \mathbb{F}_2^6$, according to the definition of $\theta(u, \gamma_i)$ in Section C.3, we have

$$\theta(u, \gamma_i) = \begin{cases} 1 & u \in \{(0,0,0,0,0,0), (0,0,0,0,1,0), (0,0,0,0,0,1)\} \\ 0 & \text{Otherwise} \end{cases}.$$

Thus, according to Equation (13), $q_i = \frac{3}{64}$ and $p_i = 0.1728421$.

Complexity and Success Probability. The output length in this application is $n = 128$. According to our DL distinguishers, the size of the set $\mathbb{D}$ of input differences is $s = |\mathbb{D}| = 63$, so $\rho = \frac{1}{64}(1+\sum_{i=0}^{62} p_i) \approx 2^{-2.43}$ and $\tau = 2^{\log(s+1)-n} = 2^{-122}$. The expectation of $|\mathbb{S}_{x,\mathbb{D}}| = \sum_{i=0}^{62} q_i$ is about $2^{1.56}$. Thus, we let $N = (\rho\tau)^{-1} = 2^{124.16}$ to make the success probability of this attack be 0.63. The time complexity of the attack can be estimated as $N(1+2^{1.98}) = 2^{124.43} \times (1+2^{1.56}) \approx 2^{126.41}$ evaluations of 4-round `Ascon` permutation.

In our attack, the selection of the $N = 2^{124.43}$ translations can be optimized by randomly choosing, e.g., $2^{104.43}$ (M_0, M_1, M_2, M_3) and under each chosen (M_0, M_1, M_2, M_3) we choose 2^{20} M_4 randomly. With this technique, the computation of (M_0, M_1, M_2, M_3) is negligible compared to other parts. Considering that `Ascon-XOF` with a 128-bit digest requires at least 2 `Ascon` permutations. Our complexity can be scaled to $2^{125.41}$ 4-round `Ascon-XOF` operations. The memory cost is negligible. Compared with the strictest approach, there are $2^{0.06}$ improved.

K Preimage Attacks on 3-Round Ascon-XOF

Notations for DL Distinguishers of the `Ascon` Permutation. Considering that the DL distinguishers will be used in analyzing the `Ascon-XOF`, the input differences and output masks of the DL distinguishers for the `Ascon` permutation in this paper are active only in the first word, and only those with 1 or 2 active bits are considered. As a result, we denote the differences and masks by the colunmn indices of their active bits. For example, (0) means an input difference or an output mask that is active in the first bit of the first word.

DL Distinguishers of `Ascon` Permutation. We apply the method in [LLL21] to the 3-round `Ascon` permutation. There are two types of DL distinguishers are considered for our preimage attack on the 3-round `Ascon-XOF`.

– Type-1 DL distinguishers:

$$\delta_i = (i), \mathbb{M}_i = \begin{Bmatrix} (i+2), (i+9), (i+12), (i+15), (i+21), \\ (i+22), (i+30), (i+31), (i+32), (i+33), \\ (i+37), (i+43), (i+44), (i+50), (i+52), \\ (i+53), (i+54), (i+55), (i+57), (i+59), \\ (i+63) \end{Bmatrix} \quad 0 \le i < 64 \tag{22}$$

The correlation of all DL distinguishers $(\delta_i, \lambda_{i,j})$, $\lambda_{i,j} \in \mathbb{M}_i$ are 1, where $\ell_i = |\mathbb{M}_i| = 21$.

– Type-2 DL distinguishers:

$$\Delta_{i+64} = (i, i+22), \mathbb{M}_{i+64} = \begin{Bmatrix} (i+2), (i+12), (i+15), (i+21), \\ (i+31), (i+37), (i+43), (i+44), \\ (i+52), (i+53), (i+54), (i+55), \\ (i+59), (i+62) \end{Bmatrix} \quad 0 \le i < 64 \tag{23}$$

The correlation of all DL distinguishers $(\delta_{i+64}, \lambda_{64+i,j})$, $\lambda_{64+i,j} \in \mathbb{M}_{i+64}$ are 1, where $\ell_{64+i} = |\mathbb{M}_{64+i}| = 14$.

We note that the padding rule of the `Ascon-XOF` is to append at least 1 bit "1" to the message, thus the last bit of messages cannot have an active difference. Hence, $\Delta_i, i \in \{63, 105, 127\}$ cannot be used in our attack. Finally, we will have 125 deterministic DL distinguishers in hand.

Let $\mathbb{D} = \{\delta_i : i \in \{0, 1, \ldots, 127\} \backslash \{63, 105, 127\}\}$, $\mathbb{M}_i = \{\lambda_{i,j} : 0 \le i < 128, 0 \le j < \ell_i\}$ derived from Equations (22) and (23). Then the input differences and output masks of the distinguishers used in our preimage attack are specified by these $\mathbb{D}$ and $\mathbb{M}_i$.

The preimage attack is similar to Section 6, where 5 message blocks are used, denoted by $(M_0, M_1, M_2, M_3, M_4) \in \mathbb{F}_2^{64\times 5}$. The 128-bit output consists of two blocks, denoted by $(T_0, T_1) \in \mathbb{F}_2^{64\times 2}$. Thus, with a direct application of Algorithm 1, we can recover a preimage of a given (T_0, T_1), as we did for the 4-round `Ascon-XOF`.

Complexity and Success Probability. The digest length is $n = 128$. The number of differences in $\mathbb{D}$ is $s = 125$, so $\rho = 1$ and $\tau = 2^{\log(s+1)-n} = 2^{-121.02}$. The expectation of $|\mathbb{S}_{x,\mathbb{D}}| = \sum_i 2^{-\ell_i} \approx 2^{-8.03}$. To make the success probability be 0.63, we let $N = (\rho\tau)^{-1} = 2^{121.02}$. The time complexity is $N\,(1 + |\mathbb{S}_{x,\mathbb{D}}|) = 2^{121.02} \times (1 + 2^{-8.83}) \approx 2^{121.02}$ 4-round `Ascon` permutations.

Although our attack uses 5-block messages, we can select the last block under each fixed first 4 blocks. Thus, the cost for computing the first 4 blocks can be much smaller than that for the last block. Considering that `Ascon-XOF` with a

128-bit digest requires at least calculating 2 `Ascon` permutations. Our complexity can be scaled to $2^{120.02}$ `Ascon-XOF` conductions. The memory cost is negligible.

L MitM-DL Preimage Attack on 3-Round `Ascon-HASH`

Similar to the 4-round attack 7, here we also focus on the recovery of S_{Tc}. We again apply the method in [LLL21] to the 3-round `Ascon` permutation. We derive 64 groups of DL distinguishers for the 3-round `Ascon-HASH`. The ith ($0 \leq i < 64$) group of the DL distinguishers are as follows,

$$\delta_{64,i} = (64+i), \mathbb{M}_{64,i} = \begin{Bmatrix} (i+9),(i+15),(i+21),(i+22),(i+30), \\ (i+31),(i+32),(i+33),(i+37),(i+43), \\ (i+44),(i+50),(i+52),(i+53),(i+54), \\ (i+55),(i+57),(i+59),(i+63) \end{Bmatrix}$$

The correlation of all DL distinguishers $(\delta_{64,i}, \lambda_{i,j})$ and $\lambda_{i,j} \in \mathbb{M}_{64,i}$ are 1, where $\ell_{64,i} = |\mathbb{M}_{64,i}| = 19$. Thus, we will use the set of difference $\mathbb{D} = \{\delta_{64,i} : 0 \leq i < 64\}$, and then $|\hat{\mathbb{D}}| = 65$.

As we will see later, we need $N = 2^{192-\log(65)}$ disjoint translations of $\hat{\mathbb{D}}$. If we directly apply Lemma 2, the space we choose these disjoint translations should be significantly larger than 2^{384}, which cannot be satisfied because the capacity part of S_T is with a size of only 2^{256}. Therefore, we choose translations of $\hat{\mathbb{D}}$ as $x \oplus \hat{\mathbb{D}}$ where $x \in \langle \hat{\mathbb{D}} \rangle$. According to Lemma 1, these translations must be disjoint.

Note that the differences $\delta_{64,i}, 0 \leq i < 64$ are all active in the second word of S_T, thus if we choose $N = 2^{192-\log(65)}$ values that the second word is zero (the nonzero bits are only possible in the third, forth and fifth rows), we can choose $2^{192-\log(65)}$ disjoint translations of $\hat{\mathbb{D}}$. Hence, we can apply Algorithm 1 to recover S_T, the process is very similar to the previous applications.

Complexity and Success Probability. In the case of recovering S_{Tc}, $n = 192$, $s = |\mathbb{D}| = 64$. So $\rho = 1$ and $\tau = 2^{\log(s+1)-n} = 2^{-185.98}$. On average, $|\mathbb{S}_{x,\mathbb{D}}| = 2^{-15}$. We let $N = (\rho\tau)^{-1} = 2^{185.98}$ to make the success probability be 0.63. The complexity is $N(1 + |\mathbb{S}_{x,\mathbb{D}}|) = 2^{185.98} \times (1 + 2^{-15}) \approx 2^{185.98}$.

To recover S_{Tc}, we need to perform the `Ascon` permutation that follows S_T. Only when T_1 is matched, we continue to check if T_2 and T_3 are also matched. Therefore, the computation for T_2 and T_3 are small. Considering that `Ascon-HASH` performs at least 4 permutations, the main part of our calculation is 1/4 of the `Ascon-HASH` computations. Thus, the complexity for recovering S_{Tc} is about $\mathsf{T}/4 \approx 2^{183.98}$.

When a S_T is recovered, we proceed with the internal collision phase. When using 5 messages, the 256-bit internal collision with two 2^{128} sets ($\mathbb{T}_{S_L}$ and $\mathbb{T}_{S_R}$) has a birthday successful probability. Considering that the collision phase costs around 2^{128} computations, which is negligible compared to the recovery of S_{Tc} phase, we can actually trade some time and memory complexity with the successful probability. For example, we can use 7 message blocks, and make

$\mathbb{T}_{S_L}$ and $\mathbb{T}_{S_R}$ have sizes of, e.g., 2^{130}, then the successful probability of the internal collision phase will boost to extremely close to 1. Further, with Floyd's cycle-finding algorithm [Flo67, Sas14], the internal-collision phase can require a negligible memory cost, so the internal collision phase can be made memoryless. Hence, the time complexity and the successful probability are as the same as the recovery of S_{Tc}.

M The trivial attack for 3.5 Steps `Schwaemm256-128`

In this section, we will introduce the structural attack for 3.5-step `Schwaemm256-128`. Firstly, we randomly chose a nonce $\boldsymbol{n}$, then call the 3.5-step `Schwaemm256-128` initialization oracle with $\boldsymbol{n}$ to encrypt a plaintext $\boldsymbol{p}$. From the resulting ciphertext $\boldsymbol{z}$ and $\boldsymbol{p}$, we can deduce the value $\boldsymbol{c} = (c_0, c_1, c_2, c_3)$ for $C = (C_0, C_1, C_2, C_3)$. As shown in Fig. 7, we can know X_0^2 from C_3 due to $C_3 = A_{c_3}(A_{c_0}(X_0^2))$. And Y_2^1 and Y_4^1 are known from the nonce $\boldsymbol{n}$. If we obtain the intermediate value Y_0^1 and Y_1^1, we can get the Key (K_0, K_1).
According to

$$X_0^2 = Y_4^1 \oplus Y_1^1 \oplus \ell(Y_0^1 \oplus Y_1^1 \oplus Y_2^1),$$

where ℓ the the linear operation inside $\mathcal{M}_3$, we can get a linear system with 128 variables and 64 equations. Then, we can get 2^{64} possible values for (Y_0^1, Y_1^1). Corresponding 2^{64} key candidates can be computed backwards by the (Y_0^1, Y_1^1). By testing these 2^{64} possible keys with c_0, c_1 and c_2, we can find the correct (K_0, K_1). The data and memory complexities are both 1 and the time complexity is about 2^{64}.

N Key-Recovery Attacks on `Schwaemm192-192`, `Schwaemm256-256` and `Schwaemm128-128`

Similar to the attack on `Schwaemm256-128` we have exemplified in Section 8, this section presents key-recovery attacks on 3.5- and 4.5-step `Schwaemm192-192`, `Schwaemm256-256` and `Schwaemm128-128`.

The notations are similar to those in Section 8. $(N_0, \dots, N_{h_1-1}) \in \mathbb{F}_2^{64h_1}$ is the $64h_1$-bit nonce, $(K_0, \dots, K_{h_2-1}) \in \mathbb{F}_2^{64h_2}$ is the $64h_2$-bit key, and $(C_0, \dots, C_{h_1-1}) \in \mathbb{F}_2^{64h_1}$ is the $64h_1$-bit output known to the attackers. For the convenience of description of the attacks, the input and output of the j-th step of the `Sparkle` permutation are denoted by $X^j = (X_0^j, \dots, X_{h_1+h_2-1}^j)$ and $Y^j = (Y_0^j, \dots, Y_{h_1+h_2-1}^j)$, respectively.

N.1 Key-Recovery Attack on `Schwaemm192-192`

In this subsection, we apply Algorithm 5 to the 3.5-step `Schwaemm192-192` to recover all its 192-bit key. The attack can be naturally extended to 4.5 steps with the same method as Section 8.2.

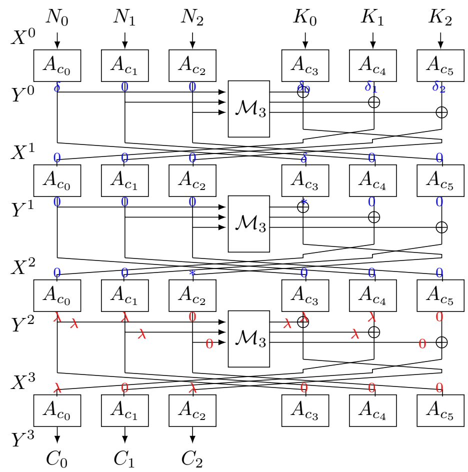


Fig. 15: The illustration of 3.5 steps of `Schwaemm`192-192 initialization. The underlying permutation is `Sparkle`384. The blue values represent the differences whereas the red values are masks.

Our strategy is to apply Algorithm 5 to the function F_{LSLSL} mapping Y^0 to X^3 to recover (Y_3^0, Y_4^0, Y_5^0). We first introduce the DL distinguishers used for F_{LSLSL}. As shown in Figure 15, let $\Lambda(X^3) = (\lambda, 0, \lambda, 0, 0, 0)$ with $\lambda \in \mathbb{F}_2^{64}\backslash\{0\}$ be the linear mask of X^3. The consequent linear mask of Y^2 is $\Lambda(Y^2) = (\lambda, \lambda, 0, \lambda, \lambda, 0)$. We set the difference of Y^0 to be $\Delta(Y^0) = (\delta, 0, 0, \delta_0, \delta_1, \delta_2)$ with $\delta \in \mathbb{F}_2^{64}\backslash\{0\}$ and $(\delta_0, \delta_1, \delta_2) = \mathcal{M}_3(\delta, 0, 0)$. According to Lemma 3, the difference of X^1 is $\Delta(X^1) = (0, 0, 0, \delta, 0, 0)$, and thus the difference of X^2 is $\Delta(X^2) = (0, 0, *, 0, 0, 0)$, where $*$ can be any nonzero value. Since $\Delta(X_0^2) = \Delta(X_1^2) = 0$, for any nonzero δ and nonzero λ, $\lambda \cdot (\Delta(X_0^3) \oplus \Delta(X_2^3)) = 0$ holds with certainty. In the application of Algorithm 5 (with necessary tweaks), (Y_3^0, Y_4^0, Y_5^0) and (Y_0^0, Y_1^0, Y_2^0) respectively play the roles of the key and the plaintext, $\mathbb{D} = \{(\delta, 0, 0, \delta_0, \delta_1, \delta_2) : \delta \in \mathbb{F}_2^{64}\backslash\{0\}, (\delta_0, \delta_1, \delta_2) = \mathcal{M}_3(\delta, 0, 0)\ \}$, $\mathbb{D}_K = \{(\delta_0, \delta_1, \delta_2) : (\delta_0, \delta_1, \delta_2) = \mathcal{M}_3(\delta, 0, 0), \delta \in \mathbb{F}_2^{64}\backslash\{0\}\}$, $\hat{\mathbb{D}}_K = \{(\delta_0, \delta_1, \delta_2) : (\delta_0, \delta_1, \delta_2) = \mathcal{M}_3(\delta, 0, 0), \delta \in \mathbb{F}_2^{64}\}$, and the sets of masks for all difference can be the same $\mathbb{M}$ is a set of 64 bases of all $\Lambda(X^3)$. For example, we use $\mathbb{M} = \{(e_i, 0, e_i, 0, 0, 0) : 0 \leq i < 64\}$ where e_i is the i-th unit vector in $\mathbb{F}_2^{64}$.

In the attack, we randomly choose a value $\boldsymbol{y} = (y_0, y_1, y_2)$ for (Y_0^0, Y_1^0, Y_2^0), invert it to obtain the corresponding nonce $\boldsymbol{n} = (n_0, n_1, n_2)$, and query the `Schwaemm`192-192 initialization oracle with the nonce $\boldsymbol{n}$ to encrypt a plaintext $\boldsymbol{p}$. From the resulting ciphertext $\boldsymbol{z}$ and $\boldsymbol{p}$, we can deduce the value $\boldsymbol{c} = (c_0, c_1, c_2)$ for $C = (C_0, C_1, C_2)$. Inverting $\boldsymbol{c}$ we get the value $\boldsymbol{x} = (x_0, x_1, x_2)$ for (X_0^3, X_1^3, X_2^3). Next, for every $\delta \in \mathbb{F}_2^{64}\backslash\{0\}$, we choose $\boldsymbol{y}_\delta = (y_0, y_1, y_2)_\delta = \boldsymbol{y} \oplus (\delta, 0, 0)$ for (Y_0^0, Y_1^0, Y_2^0), and invert it to obtain $\boldsymbol{n}_\delta$. With the encryption oracle we can get $\boldsymbol{x}_\delta = (x_0, x_1, x_2)_\delta = (x_{0,\delta}, x_{1,\delta}, x_{2,\delta})$ for (X_0^3, X_1^3, X_2^3).

Then, for each $\boldsymbol{v} = (v_0, v_1, v_2) \in \hat{\mathbb{D}}_K^{\dashv}$, we guess the value of (Y_3^0, Y_4^0, Y_5^0) to be $\boldsymbol{v}$. Compute $F_{LSLSL}(\boldsymbol{y}, \boldsymbol{v})$, and set $\boldsymbol{w} = (w_0, w_1, w_2)$ be the first three 64-bit words of $F_{LSLSL}(\boldsymbol{y}, \boldsymbol{v})$. If $\boldsymbol{w} = \boldsymbol{x}$, $\boldsymbol{v}$ is a candidate for (Y_4^0, Y_5^0), and we can confirm its correctness by using additional data.

If $\boldsymbol{v}$ is not a candidate for (Y_3^0, Y_4^0, Y_5^0) (i.e., $\boldsymbol{w} \neq \boldsymbol{x}$) or $\boldsymbol{v}$ fails to be confirmed as the key, we use the aforementioned DL distinguishers for F_{LSLSL} to quickly filter out those $\boldsymbol{v}_\delta = \boldsymbol{v} \oplus (\delta_0, \delta_1, \delta_2) = \boldsymbol{v} \oplus \mathcal{M}_3(\delta, 0, 0)$ that cannot be the right value. According to the DL distinguisher, for any nonzero λ, if the difference of Y^0 is $\Delta(Y^0) = (\delta, 0, 0, \delta_0, \delta_1, \delta_2)$, $\lambda \cdot (\Delta(X_0^3) \oplus \Delta(X_2^3)) = 0$. We have known that w_3 is the result of $(y_0, y_1, y_2, v_0, v_1, v_2)$ and $x_{3,\delta}$ is the result of the oracle queried with $\boldsymbol{n}_\delta$. Hence, $\boldsymbol{v} \oplus \mathcal{M}_3(\delta, 0, 0)$ cannot be the right value of (Y_3^0, Y_4^0, Y_5^0) if $\lambda \cdot (w_0 \oplus x_{0,\delta} \oplus w_2 \oplus x_{2,\delta}) \neq 0$ for any nonzero λ. Equivalently, only if $\lambda \cdot (w_0 \oplus x_{0,\delta} \oplus w_2 \oplus x_{2,\delta}) = 0$ for all $\lambda \in \mathbb{F}_2^{64} \backslash \{0\}$, $\boldsymbol{v} \oplus (\delta, 0, 0)$ can be a candidate (for a wrong $\boldsymbol{v} \oplus \mathcal{M}_3(\delta, 0, 0)$, it holds with probability of 2^{-64}, which is the source of the filtering).

Note that $\lambda \cdot x = 0$ for any nonzero λ is equivalent to $\lambda_i \cdot x = 0$ for all $\lambda_i \in \mathbb{M}$ because $\mathbb{M}$ is a set of bases for all λ. Let $V(x) = (\nu_0, \nu_1, \ldots, \nu_{63})$ where $\nu_i = \lambda_i \cdot x$ and $\lambda_i \in \mathbb{M}$ (note that the last three elements of $\lambda_i \in \mathbb{M}$ are always zero). To check if $\lambda \cdot (w_0 \oplus x_{0,\delta} \oplus w_2 \oplus x_{2,\delta}) = 0$ for all nonzero λ is equivalent to check if $V(\boldsymbol{x}) = V(\boldsymbol{w})$. Therefore, we can use a hash table to quickly find the collision by storing $V(\boldsymbol{x})$, and check if $V(\boldsymbol{w})$ in the table. This process is a general case of what we did in Section 8.

Complexity and Success Probability. In the data preparation phase, we use 2^{64} nonces, and invert the output by one nonlinear layer, so the time complexity is approximately $2^{64} + 2/4 \times 2^{64}$ `Schwaemm192-192` initializations. In the guessing phase, the whole key space is divided into 2^{128} translations. Processing each translation requires 1 conduction of F_{LSLSL} and 1 table-lookup. On average, there is one $\boldsymbol{v} \oplus (\delta_0, \delta_1, \delta_2)$ that can pass the 64-bit filter. Thus, the guessing phase is dominated by the 2^{129} conductions of F_{LSLSL}. Since F_{LSLSL} contains 2 nonlinear layer, so its cost can be regarded as 2/4 of the 3.5-step `Schwaemm192-192` operation. Finally, the whole time complexity is about 2^{129} `Schwaemm192-192` operations. The data complexity is obviously 2^{64} nonces. The memory complexity is to store H, which is about 2^{64} 192-bit blocks. Since all DL distinguishers in this application is deterministic, the success probability of recovering it is 1, according to Equation (18).

Extension to 4.5 Steps. With the same strategy as Section 8.2, we can prepend a round to the 3.5-step attack to extend it to a 4.5-step one. The final time and data complexity remains almost the same. But the success probability decrease to 0.63.

N.2 Key-Recovery Attack on 3.5-Step `Schwaemm256-256`

In this subsection, we apply Algorithm 5 to the 3.5- and 4.5-step `Schwaemm256-256` to recover all its 256-bit key.

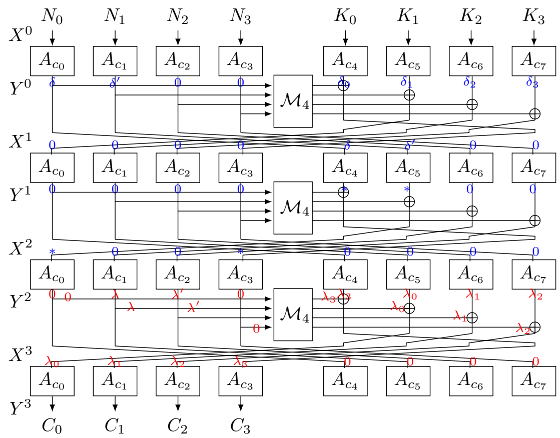


Fig. 16: The illustration of 3.5 steps of `Schwaemm256-256`. The underlying permutation is `Sparkle512`. The blue values represent the differences whereas the red values are masks.

Our strategy is to apply Algorithm 5 to the function F_{LSLSL} mapping Y^0 to X^3 to recover $(Y_4^0, Y_5^0, Y_6^0, Y_7^0)$. We first introduce the DL distinguishers used for F_{LSLSL}. As shown in Figure 16, let $\Lambda(X^3) = (\lambda_0, \lambda_1, \lambda_2, \lambda_3, 0, 0, 0, 0)$ with $(\lambda, \lambda') \in \mathbb{F}_2^{128} \backslash \{0\}$ and $(\lambda_0, \lambda_1, \lambda_2, \lambda_3) = \mathcal{M}_4(0, \lambda, \lambda', 0)$ be the linear mask of X^3. The consequent linear mask of Y^2 is $\Lambda(Y^2) = (0, \lambda, \lambda', 0, \lambda_3, \lambda_0, \lambda_1, \lambda_2)$. We set the difference of Y^0 to be $\Delta(Y^0) = (\delta, \delta', 0, 0, \delta_0, \delta_1, \delta_2, \delta_3)$ with $(\delta, \delta') \in \mathbb{F}_2^{128} \backslash \{0\}$ and $(\delta_0, \delta_1, \delta_2, \delta_3) = \mathcal{M}_4(\delta, \delta', 0, 0)$. The difference of X^1 is $\Delta(X^1) = (0, 0, 0, 0, \delta, \delta', 0, 0)$, and thus the difference of X^2 is $\Delta(X^2) = (*, 0, 0, *, 0, 0, 0, 0)$, where $*$ can be any nonzero value. Since $\Delta(X_1^2) = \Delta(X_2^2) = 0$, for any nonzero (δ, δ') and nonzero (λ, λ'),

$$(\mathcal{M}_4^{-1})^T(0, \lambda, \lambda', 0) \cdot (\Delta(X_0^3), \Delta(X_1^3), \Delta(X_2^3), \Delta(X_3^3)) = 0$$

holds with certainty (for simplicity, we use $\mathcal{M}_4$ as its corresponding matrix here). In the application of Algorithm 5 (with necessary tweaks), $(Y_4^0, Y_5^0, Y_6^0, Y_7^0)$ and $(Y_0^0, Y_1^0, Y_2^0, Y_3^0)$ respectively play the roles of the key and the plaintext, $\mathbb{D} = \{(\delta, \delta', 0, 0, \delta_0, \delta_1, \delta_2, \delta_3) : (\delta, \delta') \in \mathbb{F}_2^{128} \backslash \{0\}, (\delta_0, \delta_1, \delta_2, \delta_3) = \mathcal{M}_4(\delta, \delta', 0, 0) \}$, $\mathbb{D}_K = \{(\delta_0, \delta_1, \delta_2, \delta_3) : (\delta_0, \delta_1, \delta_2, \delta_3) = \mathcal{M}_4(\delta, \delta', 0, 0), (\delta, \delta') \in \mathbb{F}_2^{128} \backslash \{0\}\}$, $\hat{\mathbb{D}}_K = \{(\delta_0, \delta_1, \delta_2, \delta_3) : (\delta_0, \delta_1, \delta_2, \delta_3) = \mathcal{M}_4(\delta, \delta', 0, 0), (\delta, \delta') \in \mathbb{F}_2^{128}\}$, and $\mathbb{M}_i$ is a set of 128 bases of all $\Lambda(X^3)$. Note that the last four elements of $\Lambda(X^3)$ are always zero, so we actually can focus only on $\Lambda((X_0^3, X_1^3, X_2^3, X_3^3))$. Since $\Lambda((Y_0^2, Y_1^2, Y_2^2, Y_3^2)) = (0, \lambda, \lambda', 0)$, $\Lambda((Y_0^2, Y_1^2, Y_2^2, Y_3^2))$ has a set of bases with $\{(0, e_i, 0, 0) : 0 \le i < 64\} \bigcup \{(0, 0, e_j, 0) : 0 \le j < 64\}$ where e_i and e_j is the unit vector of $\mathbb{F}_2^{64}$. Thus, a set of bases of $\Lambda((X_0^3, X_1^3, X_2^3, X_3^3))$ can be $\mathbb{M} = \{(\mathcal{M}_4^{-1})^T(0, e_i, 0, 0) : 0 \le i < 64\} \bigcup \{(\mathcal{M}_4^{-1})^T(0, 0, e_j, 0) : 0 \le j < 64\}$.

In the attack, we randomly choose a value $\boldsymbol{y} = (y_0, y_1, y_2, y_3)$ for $(Y_0^0, Y_1^0, Y_2^0, Y_3^0)$, invert it to obtain the corresponding nonce $\boldsymbol{n} = (n_0, n_1, n_2, n_3)$, and query the

Schwaemm256-256 initialization oracle with the nonce $\boldsymbol{n}$ to encrypt a plaintext $\boldsymbol{p}$. From the resulting ciphertext $\boldsymbol{z}$ and $\boldsymbol{p}$, we can deduce the value $\boldsymbol{c} = (c_0, c_1, c_2, c_3)$ for $C = (C_0, C_1, C_2, C_3)$. Inverting $\boldsymbol{c}$ we get the value $\boldsymbol{x} = (x_0, x_1, x_2, x_3)$ for $(X_0^3, X_1^3, X_2^3, X_3^3)$. Next, for every $\boldsymbol{\delta} = (\delta, \delta') \in \mathbb{F}_2^{128} \backslash \{0\}$, we choose $\boldsymbol{y_\delta} = (y_0, y_1, y_2, y_3)_{\boldsymbol{\delta}} = \boldsymbol{y} \oplus (\delta, \delta', 0, 0)$ for $(Y_0^0, Y_1^0, Y_2^0, Y_3^0)$, and invert it to obtain $\boldsymbol{n_\delta}$. With the encryption oracle we can get $\boldsymbol{x_\delta} = (x_0, x_1, x_2, x_3)_{\boldsymbol{\delta}} = (x_{0,\boldsymbol{\delta}}, x_{1,\boldsymbol{\delta}}, x_{2,\boldsymbol{\delta}}, x_{3,\boldsymbol{\delta}})$ for $(X_0^3, X_1^3, X_2^3, X_3^3)$.

Then, for each $\boldsymbol{v} = (v_0, v_1, v_2, v_3) \in \hat{\mathbb{D}}_K^{\dashv}$, we guess the value of $(Y_4^0, Y_5^0, Y_6^0, Y_7^0)$ to be $\boldsymbol{v}$. Compute $F_{LSLSL}(\boldsymbol{y}, \boldsymbol{v})$, and set $\boldsymbol{w} = (w_0, w_1, w_2, w_3)$ be the first four 64-bit words of $F_{LSLSL}(\boldsymbol{y}, \boldsymbol{v})$. If $\boldsymbol{w} = \boldsymbol{x}$, $\boldsymbol{v}$ is a candidate for $(Y_4^0, Y_5^0, Y_6^0, Y_7^0)$, and we can confirm its correctness by using additional data.

If $\boldsymbol{v}$ is not a candidate for $(Y_4^0, Y_5^0, Y_6^0, Y_7^0)$ (i.e., $\boldsymbol{w} \neq \boldsymbol{x}$) or $\boldsymbol{v}$ fails to be confirmed as the key, we use the aforementioned DL distinguishers for F_{LSLSL} to quickly filter out those $\boldsymbol{v}_\delta = \boldsymbol{v} \oplus (\delta_0, \delta_1, \delta_2, \delta_3) = \boldsymbol{v} \oplus \mathcal{M}_4(\delta, \delta', 0, 0)$ that cannot be the right value. According to the DL distinguisher, for any nonzero (λ, λ'), if the difference of Y^0 is $\Delta(Y^0) = (\delta, \delta', 0, 0, \delta_0, \delta_1, \delta_2, \delta_3)$, $(\lambda_0, \lambda_1, \lambda_2, \lambda_3) \cdot (\Delta(X_0^3), \Delta(X_1^3), \Delta(X_2^3), \Delta(X_3^3)) = 0$. We have known that (w_0, w_1, w_2, w_3) is the result of $(y_0, y_1, y_2, y_3, v_0, v_1, v_2, v_3)$ and $(x_{0,\delta}, x_{1,\delta}, x_{2,\delta}, x_{3,\delta})$ is the result of the oracle queried with $\boldsymbol{n}_\delta$. Hence, $\boldsymbol{v} \oplus (\delta_0, \delta_1, \delta_2, \delta_3)$ cannot be the right value of $(Y_4^0, Y_5^0, Y_6^0, Y_7^0)$ if $(\lambda_0, \lambda_1, \lambda_2, \lambda_3) \cdot ((w_0, w_1, w_2, w_3) \oplus (x_{0,\delta}, x_{1,\delta}, x_{2,\delta}, x_{3,\delta})) \neq 0$ for any nonzero (λ, λ'). Equivalently, only if $(\lambda_0, \lambda_1, \lambda_2, \lambda_3) \cdot ((w_0, w_1, w_2, w_3) \oplus (x_{0,\delta}, x_{1,\delta}, x_{2,\delta}, x_{3,\delta})) = 0$ for all $(\lambda, \lambda') \in \mathbb{F}_2^{128} \backslash \{0\}$, $\boldsymbol{v} \oplus (\delta, \delta', 0, 0)$ can be a candidate (for a wrong $\boldsymbol{v} \oplus (\delta, \delta', 0, 0)$, it holds with probability of 2^{-128}).

Let $V(x) = (\nu_0, \nu_1, \ldots, \nu_{63})$ where $\nu_i = \lambda_i \cdot x$ and $\lambda_i \in \mathbb{M}$. To check if $\lambda \cdot ((w_0, w_1, w_2, w_3) \oplus (x_{0,\delta}, x_{1,\delta}, x_{2,\delta}, x_{3,\delta})) = 0$ for all nonzero λ is equivalent to check if $V(\boldsymbol{x}) = V(\boldsymbol{w})$ Therefore, we can use a hash table to quickly find the collision by storing $V(\boldsymbol{x})$, and check if $V(\boldsymbol{w})$ in the table. This process is a general case of what we did in Section 8.

Complexity and Success Probability. In the data preparation phase, we call the Schwaemm256-256 initialization oracle to handle 2^{128} nonces, inverted by $\boldsymbol{y}$ and $\boldsymbol{y}_\delta$, and invert the output by one nonlinear layer, so the time complexity is dominated approximately by the $2^{128} + 2/4 \times 2^{128}$ initializations. In the guessing phase, the whole key space is divided into 2^{128} translations. Processing each translation requires 1 conduction of F_{LSLSL} and 1 table-lookup. On average, one candidate can pass the filter. Thus, the guessing phase is dominated by the 2^{129} conductions of F_{LSLSL}. Since F_{LSLSL} contains 2 nonlinear layer, so its cost can be regarded as 2/4 of the 3.5-step Schwaemm256-256 operation. Finally, the whole time complexity is about $2^{129.32}$ Schwaemm256-256 operations. The data complexity is obviously 2^{128} nonces. The memory complexity is to store H, which is about 2^{128} 256-bit blocks. Since all DL distinguishers in this application is deterministic, the success probability of recovering it is 1, according to Equation (18).

Extension to 4.5 Steps. With the same strategy as Section 8.2, we can prepend a round to the 3.5-step attack to extend it to a 4.5-step one. The final time and data complexity remains almost the same. But the success probability decrease to 0.63.

N.3 Key-Recovery Attack on 3.5-Step Schwaemm128-128

In this subsection, we apply Algorithm 5 to the 3.5- and 4.5-step Schwaemm128-128 to recover all its 128-bit key.

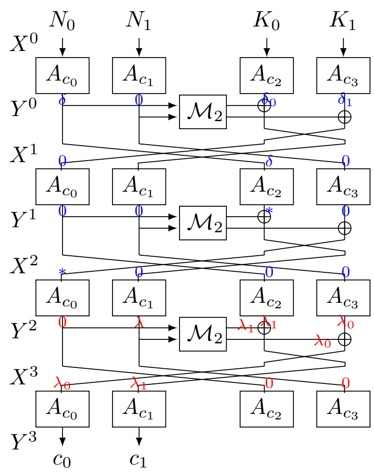


Fig. 17: The illustration of 3.5 steps of Schwaemm128-128. The underlying permutation is Sparkle256. The blue values represent the differences whereas the red values are masks.

Our strategy is to apply Algorithm 5 to the function F_{LSLSL} mapping Y^0 to X^3 to recover (Y_2^0, Y_3^0). We first introduce the DL distinguishers used for F_{LSLSL}. As shown in Figure 17, let $\Lambda(X^3) = (\lambda_0, \lambda_1, 0, 0)$ with $\lambda \in \mathbb{F}_2^{64} \backslash \{0\}$ and $(\lambda_0, \lambda_1) = \mathcal{M}_2(0, \lambda)$ be the linear mask of X^3. The consequent linear mask of Y^2 is $\Lambda(Y^2) = (0, \lambda, \lambda_1, \lambda_0)$. We set the difference of Y^0 to be $\Delta(Y^0) = (\delta, 0, \delta_0, \delta_1)$ with $\delta \in \mathbb{F}_2^{64} \backslash \{0\}$ and $(\delta_0, \delta_1) = \mathcal{M}_2(\delta, 0)$. The difference of X^1 is $\Delta(X^1) = (0, 0, \delta, 0)$, and thus the difference of X^2 is $\Delta(X^2) = (*, 0, 0, 0)$, where $*$ can be any nonzero value. Since $\Delta(X_2^2) = 0$, for any nonzero δ and nonzero λ,

$$(\mathcal{M}_2^{-1})^T(0, \lambda) \cdot (\Delta(X_0^3), \Delta(X_1^3)) = 0$$

holds with certainty (for simplicity, we use $\mathcal{M}_2$ as its corresponding matrix here). In the application of Algorithm 5 (with necessary tweaks), (Y_2^0, Y_3^0) and (Y_0^0, Y_1^0) respectively play the roles of the key and the plaintext, $\mathbb{D} = \{(\delta, 0, \delta_0, \delta_1) : \delta \in \mathbb{F}_2^{64} \backslash \{0\}, (\delta_0, \delta_1) = \mathcal{M}_2(\delta, 0)\ \}$, $\mathbb{D}_K = \{(\delta_0, \delta_1) : (\delta_0, \delta_1) = \mathcal{M}_2(\delta, 0), \delta \in \mathbb{F}_2^{64} \backslash \{0\}\}$, $\hat{\mathbb{D}}_K = \{(\delta_0, \delta_1) : (\delta_0, \delta_1) = \mathcal{M}_2(\delta, 0), \delta \in \mathbb{F}_2^{64}\}$, and $\mathbb{M}_i$ is a set of 64 bases of all $\Lambda(X^3)$. Note that the last two elements of $\Lambda(X^3)$ are always zero, so we actually can focus only on $\Lambda((X_0^3, X_1^3))$. Since $\Lambda((Y_0^2, Y_1^2)) = (0, \lambda)$, $\Lambda((Y_0^2, Y_1^2))$ has a set of bases with $\{(0, e_i) : 0 \leq i < 64\}$ where e_i is the unit vector of $\mathbb{F}_2^{64}$. Thus, a set of bases of $\Lambda((X_0^3, X_1^3))$ can be $\mathbb{M} = \{(\mathcal{M}_2^{-1})^T(0, e_i) : 0 \leq i < 64\}$.

In the attack, we randomly choose a value $\boldsymbol{y} = (y_0, y_1)$ for (Y_0^0, Y_1^0), invert it to obtain the corresponding nonce $\boldsymbol{n} = (n_0, n_1)$, and query the `Schwaemm`128-128 initialization oracle with the nonce $\boldsymbol{n}$ to encrypt a plaintext $\boldsymbol{p}$. From the resulting ciphertext $\boldsymbol{z}$ and $\boldsymbol{p}$, we can deduce the value $\boldsymbol{c} = (c_0, c_1)$ for $C = (C_0, C_1)$. Inverting $\boldsymbol{c}$ we get the value $\boldsymbol{x} = (x_0, x_1)$ for (X_0^3, X_1^3). Next, for every $\delta \in \mathbb{F}_2^{64} \backslash \{0\}$, we choose $\boldsymbol{y}_\delta = (y_0, y_1)_\delta = \boldsymbol{y} \oplus (\delta, 0)$ for (Y_0^0, Y_1^0), and invert it to obtain $\boldsymbol{n}_\delta$. With the encryption oracle we can get $\boldsymbol{x}_\delta = (x_0, x_1)_\delta = (x_{0,\delta}, x_{1,\delta})$ for (X_0^3, X_1^3).

Then, for each $\boldsymbol{v} = (v_0, v_1) \in \hat{\mathbb{D}}_K^{\dashv}$, we guess the value of (Y_2^0, Y_3^0) to be $\boldsymbol{v}$. Compute $F_{LSLSL}(\boldsymbol{y}, \boldsymbol{v})$, and set $\boldsymbol{w} = (w_0, w_1)$ be the first four 64-bit words of $F_{LSLSL}(\boldsymbol{y}, \boldsymbol{v})$. If $\boldsymbol{w} = \boldsymbol{x}$, $\boldsymbol{v}$ is a candidate for (Y_2^0, Y_3^0), and we can confirm its correctness by using additional data.

If $\boldsymbol{v}$ is not a candidate for (Y_2^0, Y_3^0) (i.e., $\boldsymbol{w} \neq \boldsymbol{x}$) or $\boldsymbol{v}$ fails to be confirmed as the key, we use the aforementioned DL distinguishers for F_{LSLSL} to quickly filter out those $\boldsymbol{v}_\delta = \boldsymbol{v} \oplus (\delta_0, \delta_1) = \boldsymbol{v} \oplus \mathcal{M}_2(\delta, 0)$ that cannot be the right value. According to the DL distinguisher, for any nonzero λ, if the difference of Y^0 is $\Delta(Y^0) = (\delta, 0, \delta_0, \delta_1)$, $(\lambda_0, \lambda_1) \cdot (\Delta(X_0^3), \Delta(X_1^3)) = 0$. We have known that (w_0, w_1) is the result of (y_0, y_1, v_0, v_1) and $(x_{0,\delta}, x_{1,\delta})$ is the result of the oracle queried with $\boldsymbol{n}_\delta$. Hence, $\boldsymbol{v} \oplus (\delta_0, \delta_1)$ cannot be the right value of (Y_2^0, Y_3^0) if $(\lambda_0, \lambda_1) \cdot ((w_0, w_1) \oplus (x_{0,\delta}, x_{1,\delta})) \neq 0$ for any nonzero λ. Equivalently, only if $(\lambda_0, \lambda_1) \cdot ((w_0, w_1) \oplus (x_{0,\delta}, x_{1,\delta})) = 0$ for all $\lambda \in \mathbb{F}_2^{64} \backslash \{0\}$, $\boldsymbol{v} \oplus (\delta, 0)$ can be a candidate (for a wrong $\boldsymbol{v} \oplus (\delta, 0)$, it holds with probability of 2^{-64}).

Let $V(x) = (\nu_0, \nu_1, \ldots, \nu_{63})$ where $\nu_i = \lambda_i \cdot x$ and $\lambda_i \in \mathbb{M}$. To check if $\lambda \cdot ((w_0, w_1) \oplus (x_{0,\delta}, x_{1,\delta})) = 0$ for all nonzero λ is equivalent to check if $V(\boldsymbol{x}) = V(\boldsymbol{w})$ Therefore, we can use a hash table to quickly find the collision by storing $V(\boldsymbol{x})$, and check if $V(\boldsymbol{w})$ in the table. This process is a general case of what we did in Section 8.

Complexity and Success Probability. In the data preparation phase, we use 2^{64} nonces, and invert the output by one nonlinear layer, so the time complexity is approximately $2^{64} + 2/4 \times 2^{64}$ `Schwaemm`128-128 initializations. In the guessing phase, the whole key space is divided into 2^{64} cosets. Processing each coset requires 1 conduction of F_{LSLSL} and 1 table-lookup. On average, there is one $\boldsymbol{v} \oplus (\delta_0, \delta_1)$ that can pass the 64-bit filter. Thus, the guessing phase is dominated by the 2^{65} conductions of F_{LSLSL}. Since F_{LSLSL} contains 2 nonlinear layer, so its cost can be regarded as $2/4$ of the 3.5-step `Schwaemm`128-128 operation. Finally, the whole time complexity is about $2^{65.32}$ `Schwaemm`128-128 operations. The data complexity is obviously 2^{64} nonces. The memory complexity is to store H, which is about 2^{64} 128-bit blocks. Since all DL distinguishers in this application is deterministic, the success probability of recovering it is 1, according to Equation (18).

Extension to 4.5 Steps. With the same strategy as Section 8.2, we can prepend a round to the 3.5-step attack to extend it to a 4.5-step one. The final time and data complexity remains almost the same. But the success probability decrease to 0.63.

O Application V: Key-Recovery Attacks on Full `Crax-S-10`

`Crax-S-10` is a lightweight block cipher built on 10 iterations of the ARX box `Alzette` [BBdS+20] whose key size and block size are 128-bit and 64-bit, respectively. As illustrated in Figure 18, the master key $K = (K_0, K_1) \in \mathbb{F}_2^{64} \times \mathbb{F}_2^{64}$ is divided into K_0 and K_1 and used alternately. In our attack, for $0 \leq i < 10$, the 64-bit input and output of A_{c_i} is written as X_i and Y_i. Thus, $Y_i \oplus K_i = X_{i+1}$. With these notations, we have $P \oplus K_0 = X_0$ and $C = K_{10} \oplus Y_9$.

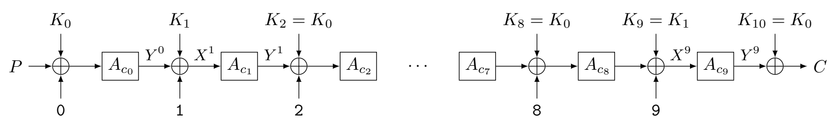


Fig. 18: The structure of `Crax-S-10`.

The basic idea of this attack is to use *deterministic* differential-linear distinguishers of A_{c_1} to bypass the A_{c_1} operation. Let (δ, λ) be a *deterministic* DL distinguisher of A_{c_1}. We set the difference of K_1 as $\Delta(K_1) = \delta$ and the mask of Y^1 as $\Lambda(Y^1) = \lambda$. Then we derive a *deterministic* related-key DL distinguisher for the second round of `Crax-S-10`, i.e., $\lambda \cdot (A_{c_1}(K_1 \oplus Y^1) \oplus A_{c_1}(K_1 \oplus Y^1 \oplus \delta)) = \zeta_{\mathtt{c}}$. Looking at the top-left corner of Table 4, each of the first 7 differences and their first 3 linear masks form three correlation-1 DL distinguishers. We denote the set of the 7 differences by $\mathbb{D}$ and it can be checked that $\hat{\mathbb{D}}$ is a linear space with the dimension being 3.

For simplicity, we define a partial decryption $D : \mathbb{F}_2^{64\times 3} \to \mathbb{F}_2^{64}, (K_0, K_1, X^9) \mapsto Y^1$. In our attack, we first call `Crax-S-10` oracle to encrypt a plaintext p and derive the corresponding ciphertext c. Then, we guess k_0 for K_0, and using k_0 to partially encrypt p to get y^0 for Y^0 and partially decrypt c to get x^9 for X^9. Next, we guess k_1 for K_1 and compute $x^1 = A_{c_1}(y^0 \oplus k_1)$ and $y^1 = D(k_0, k_1, x^9)$. Let $\gamma = A_{c_1}(x^1)$, if $\gamma = y^1$, (k_0, k_1) is a candidate for (K_0, K_1), we use another plaintext-ciphertext pair to confirm it. If $\gamma \neq y^1$ or (k_0, k_1) is not the correct key, for $\delta_i \in \mathbb{D}$, we compute $y^1_{\delta_i} = D(k_0, k_1 \oplus \delta_i, x^9)$. If $(k_0, k_1 \oplus \delta_i)$ is the correct value of (K_0, K_1), it is necessary that the equation $\lambda_{i,j} \cdot (\gamma \oplus y^1_{\delta_i}) = \lambda_{i,j} \cdot (A_{c_1}(k_1 \oplus y^0) \oplus D(k_0, k_1 \oplus \delta_i, x^9)) = \zeta_{\mathtt{c}_{i,j}}$ holds for $0 \leq j < 3$. If there is $0 \leq j_0 < 3$ making this equation invalid, $(k_0, k_1 \oplus \delta_i)$ is not the correct (K_0, K_1) for certainty. Only those passing the filters will be checked using more data.

Complexity and Success Probability. Denote the cost of one `Alzette` by T. For each guessed k_0 for K_0, we partially encrypt the plaintext and decrypt the ciphertext for one round get Y^0 and X^9, respectively, which costs $2T$. Under each guess of k_0, the key space of K_1 is divided into 2^{61} translations. For each guessed k_1, we do 8 `Alzette` operations, i.e., $8T$, to check if it is a valid candidate. If not, for each $\delta_i \in \mathbb{D}$, we perform the partial decryption with $D(k_0, k_1 \oplus \delta_i, \cdot)$, which costs $7D = 49T$ (D is a 7-round decryption). On average $7 \times 2^{-3} = 0.875$ $k_1 \oplus \delta_i$ can pass the filter, so to confirm them we need one more T for each of them. Finally, the whole time complexity is $2^{64} \times (2T + 2^{61} \times (8T + 49T + 0.875T)) = 2^{130.85}T$. Since every `Crax-S-`10 costs $10 \times T$, the time complexity of our attack is $2^{130.85-3.32} = 2^{127.53}$. The data complexity is 2 known plaintext-cipher pairs, the first pair is used in the above attack process, the second pair is used to further determine if the key candidate is the real master key (since the key length is 128, 2 plaintext-ciphertext pairs are necessary.)

Remark 6. From our the attack on `Crax-S-`10, we can see that highly biased related-key DL distinguishers over a part of the block cipher could potentially be transformed into a full-round attack. Although such transformations often lead to attacks with marginal speedup, it is interesting to see how a local weakness is transformed into a global attack. For a cipher $E = E_2 \circ E_1 \circ E_0$, suppose we have a significant enough distinguisher for E_1. To transform the distinguisher into a meaningful global attack, the data complexity for verifying the distinguisher should be low. Otherwise, since we need to partially encrypt the plaintext and decrypt the ciphertext to the input and output of E_1, the cost for the partial encryption and decryption would surpass the brute-force attack easily.