The Large Block Cipher Vistrutah

Roberto Avanzi[1], Avik Chakraborti[2], Bishwajit Chakraborty[3] and Eik List[4]

[1] Caesarea Rothschild Institute, University of Haifa, Israel
roberto.avanzi@icloud.com
[2] Institute for Advancing Intelligence, TCG Crest, Kolkata, India
avik.chakraborti@tcgcrest.org
[3] School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore
bishu.math.ynwa@gmail.com
[4] Independent Researcher
elist@posteo.net

Abstract. Vistrutah is a block cipher with block sizes of 256 and 512 bits. It iterates a *step* function consisting of two AES rounds applied to each 128-bit block of the state, followed by a state-wide cell permutation. Building upon established design principles from Simpira, Haraka, Pholkos, and ASURA, Vistrutah leverages AES instructions to achieve high performance.

For each component of Vistrutah, we conduct a systematic evaluation of functions that can be efficiently implemented on both Intel and Arm architectures. We therefore expect them to perform efficiently on any recent vector instruction set architecture (ISA) with AES support. Our evaluation methodology combines, for each combination of the various choices of the cipher's components, a security analysis with a latency estimation on an abstracted ISA. The goal is to maximize the ratio of "bits of security per unit of time," i.e., to achieve the highest security for a given performance target, or equivalently, the best performance for a given security level within this class of designs. Implementations confirm the accuracy of our latency model. Vistrutah even performs significantly better than Rijndael-256-256.

Our security claims are backed by a comprehensive ad-hoc cryptanalysis. An isomorphism between Vistrutah-512, the 512-bit wide variant, and the AES, allows us to also leverage the extensive cryptanalysis of AES and apply it to Vistrutah-512.

A core design principle is the use of an *inline* key schedule, computed during each encryption or decryption operation without requiring storage in any external memory. In fact, rekeying Vistrutah has no associated overheads. Key schedules like the AES's must precompute and store round keys in memory for acceptable performance. However, in 2010 Kamal and Youssef showed that this makes cold boot attacks significantly more effective. Vistrutah's approach minimizes leakage to at most two byte-permutations of the original key during context switches. Furthermore, expensive key schedules reduce key agility, limiting the design of modes of operation.

Vistrutah is particularly well-suited for Birthday-Bound modes of operation, including Synthetic IV modes and Accordion modes for 256-bit block ciphers. It can serve as a building block for compression functions (such as Matyas-Meyer-Oseas) in wide MerkleDamgård hash functions. Additionally, it can implement "ZIP" wide pseudo-random functions as recently proposed by Flórez-Gutiérrez et al. in 2024.

Finally, we present *short*, i.e., reduced-round versions of Vistrutah which are analyzed taking into account the restrictions posed on attackers by specific modes of operation. In particular, we model the use of the block ciphers in *Hash-Encrypt-Hash* (HEH) constructions such as HCTR2 as well as in ForkCiphers. These short versions of Vistrutah can be used to accelerate modes of operation without sacrificing security.

Keywords: Block Cipher, AES, AES Instructions, Cryptographic Implementations, Cryptanalysis.

Intellectual property rights statement. All the designs introduced in this document are placed in the public domain. The authors and their employers do not claim any intellectual property rights over the design. To the best of our knowledge no one else has any relevant intellectual property rights either.

Disclaimer. All information in this document is provided as is, and no guarantee or warranty is given or implied that the information is fit for any particular purpose. The user thereof uses the information at their sole risk and liability.

Contents

List of Tables **iv**

List of Figures **v**

1 Introduction **1**
1.1 Security Model, Levels and Claims . . . 2
1.2 Why another AES round function based large block cipher? . . . 3
1.3 Related Work . . . 4
1.4 Organization . . . 4

2 Background on CPU Instructions **4**
2.1 The AES Round Function Instructions . . . 4
2.2 Some Common Instructions on Vectors . . . 5

3 Specification of the Large Block Cipher Vistrutah **6**
3.1 The State . . . 6
3.2 Rounds, Steps, and the Design of the Cipher . . . 7
3.3 The Mixing Layer . . . 7
3.4 The Key Schedule . . . 9
3.5 The Round Constants . . . 11

4 Choice of the Components of Vistrutah **12**
4.1 A Model of the Computational Cost of a Cipher Design . . . 12
4.2 The Role of a Preliminary Differential and Linear Cryptanalysis . . . 13
4.3 Design Options and Choices for Vistrutah . . . 14
4.4 Costs of the Steps of Various Ciphers . . . 22
4.5 Final Selection of the Components . . . 22

5 Implementing the Large Block Ciphers **25**
5.1 Software Implementation . . . 25
5.2 A word about Hardware . . . 28

6 Summary of the Cryptanalysis of Vistrutah **30**
6.1 Diffusion and Structure . . . 30
6.2 Isomorphism between the AES and AESQ-like Transforms . . . 31
6.3 Differential and Linear Cryptanalysis . . . 32
6.4 Integral Cryptanalysis . . . 32
6.5 Impossible-differential Cryptanalysis . . . 32
6.6 Boomerang Cryptanalysis . . . 33
6.7 Mixture-differential Cryptanalysis . . . 34
6.8 Yoyo Cryptanalysis . . . 34
6.9 Multiple-of-n Cryptanalysis . . . 35
6.10 Demirci-Selçuk Meet-in-the-middle Cryptanalysis . . . 35
6.11 Slide Attacks . . . 36
6.12 Quantum Security . . . 36

7 Security in the Context of Practical Settings **36**
7.1 Settings . . . 36
7.2 Security of Round-reduced Vistrutah in the SPRP Setting . . . 39
7.3 Security of Round-reduced Vistrutah in the PRP Setting . . . 41
7.4 Security of Round-reduced Vistrutah in the wiPRP Setting . . . 41
7.5 Security of Round-reduced Vistrutah in the wimPRP Setting . . . 42
7.6 Security of Round-reduced Vistrutah in the wisPRP Setting . . . 42
7.7 Key Agility . . . 43

8 Conclusion **43**

References **44**

A Selected Active Cell Counts for Differential Cryptanalysis **65**

B Detailed Performance Data **68**

C Alternative Round Constants for Vistrutah **70**

D Breakdown of the Costs of the Steps for all the Ciphers **71**

E Detailed Cryptanalysis of Vistrutah **73**
E.1 Integral Cryptanalysis . . . 73
E.2 Impossible-differential Cryptanalysis . . . 85
E.3 Boomerang Cryptanalysis . . . 106
E.4 Mixture-differential Cryptanalysis . . . 119
E.5 Yoyo Cryptanalysis . . . 128
E.6 Multiple-of-n Cryptanalysis . . . 137
E.7 Demirci-Selçuk Meet-in-the-middle Cryptanalysis . . . 141

List of Tables

1 Costs of various step functions for encryption and decryption. . . . 23
2 Costs in numbers of instruction groups at which various ciphers with $s = 2$ are estimated to reach set security levels against differential attacks where the adversary can control the input. . . . 23
3 Costs in numbers of instruction groups at which various ciphers with $s = 4$ are estimated to reach set security levels against differential attacks. . . . 23
4 Performance comparisons on an Apple Silicon M3 computer. . . . 26
5 Performance comparisons on an Intel Core i7 computer. . . . 27
6 Approximate area and delay of the ciphers considered in this paper in HW. 29
7 Summary of the cryptanalysis of round-reduced Vistrutah. . . . 30
8 Estimations of the number of rounds covered by attacks in different practical modes of operation. . . . 41
9 Minimal number of active S-Boxes for differential characteristics, $s = 2$. . . 65
10 Minimal number of active S-Boxes for pure related-key differential characteristics, $s = 2$. . . . 65
11 Minimal number of active S-Boxes for single-key and pure related-key differential characteristics, $s = 2$. . . . 65
12 Minimal number of active S-Boxes for cell-wise differential characteristics, $s = 4$. . . . 66
13 Minimal number of active S-Boxes for pure related-key differential characteristics, for $s = 4$, with stretched 256-bit keys. . . . 66

14 `Vistrutah.A` and `Pholkos`, $s = 4$, with 512-bit keys: Minimal number of active S-Boxes for related-key differential characteristics. 67
15 `Vistrutah.B` and `Vistrutah.A`-3R, $s = 4$, with stretched 256-bit keys and full 512-bit keys: Minimal number of active S-Boxes for single-key and pure related-key differential characteristics. 67
16 Timings of selected versions of the ciphers considered in this paper on an Apple Silicon M3 computer. 68
17 Timings of selected versions of the ciphers considered in this paper on an Intel Core i7 computer. 69

List of Figures

1 Step and Inverse Step of `Vistrutah`-R2 with the Intel AESNI instructions. 8
2 Step and Inverse Step of `Vistrutah`-R2 with the Arm NEON instructions. 8
3 Step and Inverse Step of `Vistrutah`-R3 with the Intel AESNI instructions. 8
4 Step and Inverse Step of `Vistrutah`-R3 with the Arm NEON instructions. 8
5 `Vistrutah`-256 and `Vistrutah`-512 (even number r of rounds). 9
6 The round constants used in `Vistrutah` (given as hexadecimal values). . 12
7 Key Recovery. 13
8 Step and Inverse Step of `Vistrutah.B` with non-zero keys (using NEON). 15
9 Step and Inverse Step of `Vistrutah.B` with non-zero keys (using AESNI). 15
10 `Vistrutah.B`-256 and `Vistrutah.B`-512 ($3 \mid r$). 16
11 Computation flows in various mode-aware weak iterated input-transform PRP notions. 40
12 Code for the round constant generating function PSI (Ψ) and its inverse. 70
13 Integral distinguisher on five-round `Vistrutah`-256. 74
14 The key-recovery rounds of the seven-round integral attack on `Vistrutah`-256. 74
15 The key-recovery rounds of the eight-round integral attack on `Vistrutah`-256. 76
16 Integral attack on six-round `AES`-128. 78
17 Integral trail on eight-round `Vistrutah`-512. 79
18 The key-recovery rounds of the 12-round integral attack on `Vistrutah`-512. 80
19 The key-recovery rounds of the 12-round integral attack on `Vistrutah`-512. 83
20 Distinguisher and key-recovery phases in the 7-round impossible-differential attack on `Vistrutah`-256. 86
21 Distinguisher and key-recovery phases in the 8-round impossible-differential attack on `Vistrutah`-256. 88
22 Distinguisher and key-recovery phases in the 8-round impossible-differential attack on `Vistrutah`-256. 90
23 The impossible-differential trail in the attack on 10-round `Vistrutah`-512. 93
24 The key-recovery rounds of the 10-round impossible-differential attack on `Vistrutah`-512. 94
25 Impossible-differential trail in the attack on 10-round `Vistrutah`-512-256. 97
26 The key-recovery rounds of the 10-round impossible-differential attack on `Vistrutah`-512-256. 98
27 The first seven of the eight-round distinguisher in the impossible-differential attack on 14-round `Vistrutah`-512. 102
28 The outer rounds of the online filtering and key-recovery phase for the 14-round impossible-differential attack on `Vistrutah`-512. 103
29 Boomerang attack on eight-round `Vistrutah`-256: Forward trail. 107
30 Boomerang attack on eight-round `Vistrutah`-256: Backward trail. 108
31 The original boomerang on 6-round `AES`. 111

32 First part of the forward differential trail of the boomerang attack on 12-round Vistrutah-512, from pairs P and P' to intermediate states U. . 112
33 Lower part of the differential trail of the boomerang attack on 12-round Vistrutah-512. 113
34 Lower part of the differential trail of the boomerang attack on 12-round Vistrutah-512. 114
35 Upper part of the decryption trail of the boomerang attack on 12-round Vistrutah-512. 115
36 Upper part of the decryption trail used in the boomerang attack on 12-round Vistrutah-512-256. 118
37 The mixture attack on 8-round Vistrutah-256. 120
38 Mixture distinguisher on 10-round Vistrutah-512: Rounds 2–6. 122
39 Mixture distinguisher on 10-round Vistrutah-512: Rounds 7–11. 123
40 Mixture distinguisher on 12-round Vistrutah-512: Rounds 2–8. 125
41 Mixture distinguisher on 12-round Vistrutah-512: Rounds 9–13. 126
42 Yoyo attack on 8-round Vistrutah-256: Forward differential trail. 128
43 Yoyo attack on 8-round Vistrutah-256: Backward differential trail. . . . 129
44 Yoyo attack on 10-round Vistrutah-512: Upper forward differential trail. 132
45 Yoyo attack on 10-round Vistrutah-512: Lower forward differential trail. 133
46 Yoyo attack on 10-round Vistrutah-512: Lower backward differential trail. 134
47 Yoyo attack on 10-round Vistrutah-512: Upper backward differential trail. 135
48 Multiple-of-2 distinguisher on seven-round Vistrutah-256. 138
49 Multiple-of-8 distinguisher on 10-round Vistrutah-512: Rounds 2–6. . . 139
50 Multiple-of-8 distinguisher on 10-round Vistrutah-512: Rounds 7–11. . . 140
51 8-round DS-MitM attack on Vistrutah-256. 141
52 8-round DS-MitM attack on Vistrutah-512. 143
53 The offline distinguisher in the attack on Vistrutah-256. 146
54 The outer rounds of the online filtering and key-recovery phase for the 10-round attack on Vistrutah-256. 146
55 The offline distinguisher in the attack on 12-round Vistrutah-512. 148
56 The outer rounds of the online filtering and key-recovery phase for the 12-round attack on Vistrutah-512. 149

1 Introduction

The recent call for comments by NIST (National Institute of Standards and Technology) on 256-bit `Rijndael` variants has revived interest in block ciphers with block size of at least 256 bits. This research area was however active even before the `AES` standardization.

A significant amount of work exists on block-size doubling techniques to construct larger ciphers from smaller ones. The Luby-Rackoff (LR) construction serves as a foundational approach, providing *Chosen-plaintext Attack* (CPA) security with three rounds and *Chosen-ciphertext Attack* (CCA) security with four rounds [LR88, Pie90]. Recent quantum security analysis [GJNS14, HS18, HI19, IHM+19, XY23] shows that at least six LR rounds are required to achieve *Q1* model security, i.e., where a quantum computer analyzes classical queries. This requirement makes the approach impractical.

Alternative doubling techniques increase block size while maintaining the original key size. Methods such as Elastic [CYK04, CKY07], XLS [RR07], HEM [Zha12], and LDT [CLMP17] produce ciphers with $(2n-1)$-bit blocks. These would need double application to construct a $2n$-bit cipher. Direct $2n$-bit cipher constructions like *ECB-Mix-ECB* (EME) [HR04] and QuEME [BCF+24] require five cipher invocations while being constrained by the security of the original key size.

For better performance *and* security, designers have proposed ciphers with native large block sizes. Beside `Rijndael` [DR02], some examples are: `BassOmatic` (2048 bits, part of PGP 1.0, released in 1991); `xmx` (256, 512, 768 or 1024 bits) [MNSV97]; `Mercy` (4096 bits) [Cro00]; `NUSH` and `NASH` (64, 128, or 256 bits) [Pre01, LKK16]; `SHACAL-2` (256 bits, replacing the instance of SHA-1 in [HKR01] with SHA-2); `Threefish` (256, 512 or 1024 bits) [FLS+10]; `XXTEA` (any size of at least 64 bits) [STGD12]; `Kalyna` (128, 256 or 512 bits) [OGK+15]; `Simpira` v2 (any multiple of 128 bits) [GM16]; `Marvin` (256 bits) [SRB18]; Saturnin (256 bits) [CDL+20]; `TRAX` (256 bits) [BBdS+20]; `Pholkos` (256 and 512 bits) [BLLS22]; `Ghidle` (256 and 512 bits) [NSA+23]; and `ASURA` (256 bits) [TSS+24]. The hash function `Haraka` [KLMR16] can be used to generate a keystream 256 bits at a time, and the 512-bit permutation `AESQ` [BK14] is used in the *Authenticated Encryption* (AE) algorithm `PAEQ`. Our contribution, `Vistrutah`, is a block cipher with 256- and 512-bit blocks.

Some of these ciphers, including ours, use `AES` instructions to achieve high performance. `Simpira`, `Haraka`, and `Pholkos`, for instance, employ pairs of `AES` rounds as 128-bit keyed S-boxes. We build on this design philosophy, improving on both security and performance.

Interestingly, NIST is also considering "Accordion" ciphers, i.e., length-preserving *Variable Input Length* (VIL) *Tweakable Block Ciphers* (TBCs). These designs generally fall into two categories: those with *Birthday-bound* (BB) security, i.e., secure up to $2^{n/2}$ queries, and those with *Beyond-birthday Bound* (BBB) security, i.e., secure up to $q = 2^s$ queries where $n \geq s > n/2$. When $q = 2^n$, we say the cipher achieves *full security*. While BBB designs are attractive, they also have drawbacks compared to BB designs using large block ciphers — such as higher latency, greater implementation complexity, and more challenging design and analysis, though some, like bbb-ddd-AES [DMMT24] are efficient when sufficient parallel resources are available. Most tweakable BBB modes with a block cipher call per block achieve security only up to $2^{2n/3}$ calls. Full-security tweakable modes typically need at least two n-bit block cipher calls per block [Men15, BBD+24].

NIST's dual interests in large block ciphers — *i.e., with a block size of at least 256 bits — and accordion designs are clearly complementary. A BB accordion paired with a large block cipher can meet the security needs of both commercial and government applications for the foreseeable future.* `Vistrutah` *addresses this block cipher need, though nothing prevents its use in BBB modes.*

In modes of operation, the tweak can often be managed effectively by the higher-level construction, allowing the use of non-tweakable primitives like the `AES` as building blocks.

Therefore, we designed Vistrutah as a non-tweakable block cipher, prioritizing simplicity, efficiency, and security. Although not a lightweight design, Vistrutah delivers competitive performance compared to other AES-based ciphers and integrates seamlessly into *Strong Pseudo-Random Permutation* (SPRP) constructions after Naor and Reingold [NR99a, NR99b], such as HCTR2 [CHB21], FAST [CGLS22], and ACCOR [DFUB24].

A critical design decision is to avoid storing the round keys in Random-access Memory *(RAM), even if just temporarily — the key schedule must be lightweight and efficiently computed inline. This also paves the way for modes of operation that rely on key agility to achieve BBB security.*

This is in stark contrast to the AES and its expensive key schedule, which severely impacts performance if computed inline in software. Hence, the round keys are usually precomputed and stored in RAM. This carries security concerns: In [KY10, ZNW21], cold boot attacks [HSH+08, HSH+09, YADA17, WCJ+21] are significantly improved by exploiting relationships between the round keys to compensate for bit decay. Even keeping the round keys in the register file is insufficient, as context switches can still expose them to memory. Additionally, the AES key schedule inefficiently distributes key bit influence [LP25] and provides suboptimal resistance against *Related-Key* (RK) attacks [BKN09, BDF24].

More than two decades of block cipher research have improved our understanding of key schedules. Modern designs typically employ iterated cell permutations and "nothing-up-my-sleeve" round constants. We follow this approach with a twist, namely *we add a fixed round key in the even-numbered rounds, whereas we iteratively apply a permutation to the round key added in the odd-numbered rounds. Thus the relations between the key bits that an attacker would have to exploit to attack the cipher are different at every round.*

Regarding the drawbacks of the lack of key agility of the AES/Rijndael, we can just have a look at Counter-in-Tweak schemes by Peyrin and Seurin [PS16]. To be instantiated, these either require a construction that builds a tweakable block cipher from a non-tweakable one, or a native tweakable cipher. A RK attack resistant block cipher, with a fast key schedule and a large key space (256 or 512 bits), such as Vistrutah, paired with a suitable key update function, can also reap the benefits of such schemes, by using the key input in place of the tweak input, in other words as a tweakey input. AES-256 and Rijndael-256-256, however, would not be able to benefit from these schemes.

Another design goal is to attain best-in-class encryption performance while maintaining efficient decryption. Although decryption need not match encryption speed, it should be fast enough to support modes of operation that require decryption calls without significant performance degradation.

1.1 Security Model, Levels and Claims

Vistrutah is a bricklayer block cipher design. It alternates pairs of AES rounds applied to two or four 128-bit *slices* in parallel with a shuffle of the state bytes in order to ensure diffusion across the entire state. This structure is called a *step*.

We define versions with 256- and 512-bit security levels.

For each considered combination of the components of the design (mixing layer, key schedule, key stretching function), we determine the number of rounds required to achieve each desired security level. The final choice of components is driven by lower bounds on the complexity of differential cryptanalysis. The latter is estimated per "unit of time," not round counts, and performed both under the single-key model and under the RK model. Then, this is augmented with a very detailed cryptanalysis that covers the most important and effective attacks mounted on bricklayer block cipher constructions. Finally, for each targeted security level, we consider 20 % more rounds in the distinguishers than those required to reach the targeted security level, or, alternatively, the number of rounds required to reach 20 % more "bits of security" than the targeted level.

In practice, Vistrutah will not be used directly to encrypt data. Instead of ECB and other modes like CBC, OFB, and CFB, which may provide chosen-input encryption or decryption oracles for the block cipher, we expect that Vistrutah will be used in, say, HEH constructions such as HCTR2. These derive encryption and hashing keys from a *main key* (and a nonce), and have important properties such as: (i) The number of cipher rounds between any known input and output is up to doubled; (ii) The derived keys used in the plaintext encryption have no usable relations between each other even if the attacker can choose the *main keys*; and (iii) The actual input to the encryption functions is not directly observable by the attacker. We therefore have another look (in Section 7) at the results of the cryptanalysis in light of these properties.

We define parameters for *long* and *short* versions of Vistrutah:

- **Long Versions of Vistrutah and Security Claims.**

  The *long* versions of Vistrutah are 14-round Vistrutah-256 and Vistrutah-512-256 (i.e., Vistrutah-512 with a 256-bit key) and 18-round Vistrutah-512.

  For them we claim, respectively, 256 and 512 bits of security, i.e., any successful classical attack on the two ciphers will have a running time of at least $2^{256-\epsilon}$, resp., $2^{512-\epsilon}$ encryptions, and with a limit of $2^{256-\epsilon}$ blocks on the amount of memory and of data stored at any moment in time required for both. Here, ϵ is a small positive constant to compensate for biclique attacks [BKR11] and similar optimizations, we fix $\epsilon = 4$ for the 256-bit wide version and $\epsilon = 8$ for the 512-bit wide versions (motivated by the fact that biclique attacks on the 128-bit AES have $\epsilon = 2$).

- **Short Versions of Vistrutah and Security Claims.**

  The short versions are 10-round Vistrutah-256 and Vistrutah-512-256, and 12-round Vistrutah-512-512. These are intended for use in the encryption of HEH constructions (such as HCTR2) — and in such modes only in the encryption part — as well as in ForkCiphers [Ava13, ALP+19] and similar designs.

  For these versions, we claim (BB) security of 256, resp., 512 bits with HCTR2 or a similar mode, in the sense that at least $2^{128-\epsilon}$, resp., $2^{256-\epsilon}$ queries are necessary to break the enciphering scheme. Such modes are assumed to use hashes matching the block cipher width, the long version of the block cipher in the key derivation and direct encryption parts, and the short version for the counter mode encryption.

These definitions are supported by the cryptanalysis performed in Sections 4 and 6.

Vistrutah is easy to implement, versatile, secure and efficient. It is suitable for file and storage encryption and secure communication. The performance of Vistrutah makes it ideal to construct large compression functions, for instance a Matyas-Meyer-Oseas compression function [MMO85], to be used in prefix-free MerkleDamgård (MD) hash functions [Mer79, Mer89, Dam89, CLNY06] or chopMD hash functions [CN08]. In short versions, Vistrutah looks like an ideal primitive for constructing wide ForkCiphers or ZIP *Pseudo-random Function*s (PRFs) [FGL+24].

1.2 Why another AES round function based large block cipher?

The motivation extends beyond exploring an established design space. Vistrutah delivers superior performance while meeting all design objectives stated above. The new key schedule allows us to achieve resistance to RK attacks at a similar level to single-key attacks. Furthermore, the cipher is designed to offer *long* and *short* versions, where the latter is round-reduced to be used in keystream generation methods.

More critically, adopting Rijndael-256-256 as a standard without modifications would be problematic. Its implementation in software requires costly corrections to the ShiftRows

operation - computational effort better spent on an in-line key schedule and robust state mixing layer. Furthermore, versions of Rijndael with blocks larger than 128 bits have undergone limited cryptanalysis, while possibly inheriting the AES key schedule's weaknesses against RK attacks. These considerations drove our design of a "new Rijndael" with 256- and 512-bit block sizes. The 512-bit version, though approximately twice as slow as the 256-bit variant at equivalent security levels, offers comparable bandwidth with improved diffusion properties and a larger BB margin, strengthening modes of operation built upon it. While Pholkos already addresses some of the same concerns, its heavier key schedule remains a limitation. The cryptanalysis included in this paper may well surpass the cryptanalysis conducted on Rijndael-256-256 to date, and the security of the 512-bit wide version can essentially reuse the entire cryptanalysis conducted on the AES so far.

1.3 Related Work

Besides the many large block ciphers mentioned earlier, the work by Shiba et al. [SSI22] deserves attention. The paper studies large block ciphers that iterate a step consisting of one or two AES rounds followed by a state permutation. The authors examine a much larger space of state permutations than our study, though they focus on a more limited set exclusively built from the unpack and blend instructions on Intel architectures. Due to the large number of candidate functions, they limit their analysis to active cell counts over just 8 or 9 AES rounds, whereas we reach between 12 and 18 rounds. Furthermore, they do not include key schedules, which reduces the complexity of the analysis, but this also means that they cannot consider a RK security evaluation. They consider only the encryption direction, so they do not need to determine the implementation complexity of the inverses of their state permutations. This would otherwise be problematic for those based on Intel's unpack operations, which lack efficient inverses. While their paper offers valuable insights — including for our own research — these limitations restrict its broader applicability. In contrast, we consider both encryption and decryption, and find efficient non-trivial inverses for some state permutations on the Intel architecture.

1.4 Organization

In Section 2 we recall the definitions of the AES instructions and other vector instructions provided by the Intel and Arm architectures. The full Vistrutah specification is given in Section 3. In Section 4 we give a detailed report of the process that led to our choice of components and parameters for Vistrutah, which includes an evaluation of the *security per unit of processing time* for all considered variants, with some technical details moved to Appendix D. Implementation results are reported and discussed in Subsection 5.1. Section 6 presents a summary of our detailed cryptanalysis, again with the technical details moved to Appendix E. We interpret the cryptanalysis in several practical settings in Section 7. We conclude in Section 8.

2 Background on CPU Instructions

We detail here the *Central Processing Unit* (CPU) instructions used to implement all the ciphers in this work.

2.1 The AES Round Function Instructions

For the definition of the AES, its round function, and the four operations that compose the latter, namely AddRoundKey (ARK), MixColumns (MC), SubBytes (SB), and ShiftRows (SR), we refer to [DR02, DBN+01]. Many current *Instruction-set Architectures* (ISAs)

provide single round `AES` encryption and decryption operations. For instance, Intel's AESNI [Gue12, Int23a, Int23b] contains the following instructions

$$\texttt{AESENC} = \mathsf{ARK} \circ \mathsf{MC} \circ \mathsf{SB} \circ \mathsf{SR} \ , \qquad \texttt{AESENCLAST} = \mathsf{ARK} \circ \mathsf{SB} \circ \mathsf{SR} \ ,$$
$$\texttt{AESDEC} = \mathsf{ARK} \circ \mathsf{MC}^{-1} \circ \mathsf{SB}^{-1} \circ \mathsf{SR}^{-1} \quad \text{and} \quad \texttt{AESDECLAST} = \mathsf{ARK} \circ \mathsf{SB}^{-1} \circ \mathsf{SR}^{-1} \ ,$$

where the notation $g \circ f(x)$ means that f is applied to x first and then g is applied to $f(x)$, as well as the inverse `MixColumns` operation `AESIMC`, which must be applied to the round keys for decryption.

The Arm NEON instruction set extensions [Arm24] provide

$$\texttt{AESE} = \mathsf{SB} \circ \mathsf{SR} \circ \mathsf{ARK} \quad \text{and} \quad \texttt{AESD} = \mathsf{SB}^{-1} \circ \mathsf{SR}^{-1} \circ \mathsf{ARK} \ ,$$

as well as instructions `AESMC` and `AESIMC` for MC and its inverse, respectively. The decryption key schedule is different from the encryption key schedule, just as with the Intel `AES` extensions. For vectors up to 512 bits, SVE provides operations that compute up to four instances of entire rounds in parallel. In modern implementations of the Arm architecture, `AESE`/`AESMC` and `AESD`/`AESIMC` pairs are fused provided they are sufficiently close to each other in the code.

Many other architectures provide `AES` extensions, e.g., RISC V [MNP+21, RZBM24] and POWER [IBM18].

2.2 Some Common Instructions on Vectors

We list the operations on vector registers that we have considered to implement all the possible variants of `Vistrutah` and of the other ciphers compared here. We assume that logical operations such as `AND`, `OR`, `XOR`, `NOT` are available, as well as shifts and rotations by the same amount of bits in parallel on all bytes, words or double words of the vector registers. What remains to discuss, then, are instructions used to implement permutations of the bytes in a vector register. These are as follows:

1. *Arbitrary permutations within slices or across more slices.*

   The Arm NEON instruction set operates on 128-bit registers. Its table lookup instructions work as follows: a source vector provides 16 byte-sized indices into a table of 16 to 64 bytes stored in one to four source vectors; the selected bytes are placed in corresponding positions of the destination vector; and out-of-bound indices result in zero values. SVE offers this capability for up to two source registers.

   The x86-64 architecture handles table lookups differently: AVX supports byte-wise shuffles within 128-bit vectors; AVX2 enables byte-wise shuffles in parallel inside each half of 256-bit registers; AVX2's `VPERMD` instruction allows arbitrary 32-bit word selection within 256-bit vectors; Full 32 byte shuffles require combining permutations with blending and extraction; Only AVX512 (with the VBMI extension) supports complete table lookups across one or two 512-bit registers.

   For byte selection between two registers, pre-AVX512 Intel architectures must rely on `blend` instructions. In a `blend` instruction the bits of a mask are used to select whether an element of the destination vector is taken from the element in the same position in the first or second source register. Arbitrary permutations between two or more registers are performed by a combination of `blend` and `shuffle` instructions.

2. *Byte-wise rotation and alignment instructions.*

   These operations are performed in NEON using `VEXT` and in AVX using `PALIGNR` (actually a SSSE3 instruction).

3. *Arm's ZIP, UZP, and transpose operations, Intel's* ***unpack****.*

   NEON provides the ZIP instructions to interleave the elements of two vectors, for instance If $a = [a_0, a_1, a_2, a_3, \ldots, a_{15}]$ and $b = [b_0, b_1, b_2, b_3, \ldots, b_{15}]$, then

$$\begin{cases} \texttt{ZIP1}(a,b) = [a_0, b_0, a_1, b_1, \ldots, a_7, b_7] \\ \texttt{ZIP2}(a,b) = [a_8, b_8, a_9, b_9, \ldots, a_{15}, b_{15}] \end{cases} .$$

   ZIP1/ZIP2 can operate at the granularity of byte, short word, long word, and quadword. The equivalent Intel instruction pair is unpacklo/unpackhi. While NEON provides the inverse of ZIP in the form of the UZP1/UZP2 instruction pair, under AVX it must be emulated by two pairs of shuffle operations, the second of which reorders half-registers from two source vectors.

   NEON also provides operations to transpose elements in two source vectors at a granularity of byte, short word, long word, and quadword: If $a = [a_0, a_1, a_2, a_3, \ldots]$ and $b = [b_0, b_1, b_2, b_3, \ldots]$ (written out in little endian representation), then

$$\begin{cases} \texttt{TRN1}(a,b) = [a_0, b_0, a_2, b_2, \ldots] \\ \texttt{TRN2}(a,b) = [a_1, b_1, a_3, b_3, \ldots] \end{cases} \text{ i.e. } \begin{bmatrix} a_{2i} & a_{2i+1} \\ b_{2i} & b_{2i+1} \end{bmatrix} \mapsto \begin{bmatrix} a_{2i} & b_{2i} \\ a_{2i+1} & b_{2i+1} \end{bmatrix} \text{ for all } i .$$

   With AVX, naïvely one would perform a shuffle, a blend, and another shuffle. However, the operations that use transposes in this paper can be simplified, for instance one can use the unpack instructions creatively, as shown in (13).

The timing characteristics of these instructions vary across architectures and microarchitectures. In-lane permutations, Arm's ZIP/UZP and transpose operations, and Intel's unpack, blend, and shuffle instructions typically have low latency and high throughput, and are available on all ports of the vector unit. Complex cross-lane operations incur significantly higher costs. Therefore, they should only be used when no faster alternatives exist. For detailed performance characteristics of these instructions on popular Arm and Intel microarchitectures, see [Joh23, Int23a, Arm23a, Arm23b]. Similar performance behavior is expected for implementations of the RISC-V vector extensions.

3 Specification of the Large Block Cipher Vistrutah

In the following, we give a precise specification of Vistrutah.

3.1 The State

A state in Vistrutah consists of $s = 2$ or 4 *slices*. Each slice is a 128-bit AES state, represented as a 4×4 matrix of *cells* in *column-major ordering* as in [DR02], i.e.

$$\begin{bmatrix} c_{0+16k} & c_{4+16k} & c_{8+16k} & c_{12+16k} \\ c_{1+16k} & c_{5+16k} & c_{9+16k} & c_{13+16k} \\ c_{2+16k} & c_{6+16k} & c_{10+16k} & c_{14+16k} \\ c_{3+16k} & c_{7+16k} & c_{11+16k} & c_{15+16k} \end{bmatrix} = \begin{bmatrix} c_{0,0,k} & c_{0,1,k} & c_{0,2,k} & c_{0,3,k} \\ c_{1,0,k} & c_{1,1,k} & c_{1,2,k} & c_{1,3,k} \\ c_{2,0,k} & c_{2,1,k} & c_{2,2,k} & c_{2,3,k} \\ c_{3,0,k} & c_{3,1,k} & c_{3,2,k} & c_{3,3,k} \end{bmatrix} \tag{1}$$

where the three indices (i, j, k) identify a cell by its *row*, *column* and slice, and $0 \le k < s$. The state is thus $128\,s$ bits wide. We assume that the memory layout is little endian, so if the address in memory of Cell 0 is a, then Cells 4, 16, and 63 are stored at addresses $a + 4$, $a + 16$, and $a + 63$, respectively. As a vector, a slice is given as

$$[\, c_{0+16k},\, c_{1+16k},\, c_{2+16k},\, c_{3+16k},\, c_{4+16k},\, \ldots\, c_{14+16k},\, c_{15+16k}\,] .$$

Columns are also numbered from 0 to 15, whereby columns 0 to 3 of slice i are numbered from $4\,i$ to $4\,i+3$. In accordance with common terminology for tensors, the vectors of cells perpendicular to the slices, i.e., $[c_a, c_{a+16}]$ or $[c_a, c_{a+16}, c_{a+32}, c_{a+48}]$ are called *tubes*.

Keys and round keys adopt the same terminology, therefore we can talk about the cells, slices, columns, rows, and tubes of keys and round keys.

3.2 Rounds, Steps, and the Design of the Cipher

The fundamental building block of `Vistrutah` is the *step*. `Vistrutah` is an iterated block cipher consisting of homogeneous steps, except for the last step.

Definition 1. By *round* we always understand an `AES` round, either full or final, as well as the parallel application of an `AES` round function to multiple state slices.

A *step* consists of r_{st} chained `AES` rounds computed in parallel on the s 128-bit slices of the state, followed by a linear mixing layer (except for the last step) to provide diffusion across the slices of the state. The total number of rounds thus equals the number of rounds per step times the number of steps.

A step of `Vistrutah` only contains full `AES` rounds, except for the *last* round of the last step of the cipher, which omits `MixColumns`. For brevity, the cipher variants with $r_{st} = 2$, resp., $r_{st} = 3$, are denoted by the notations 2R, resp., 3R, if necessary to avoid confusion. Ultimately, we have opted for a 2R design, the rationale being explained in Subsubsection 4.3.1 and Subsection 4.5.

The operations in a 2R `Vistrutah` step are illustrated in Figures 1 and 2, and a 3R variant is depicted in Figures 3 and 4. This results in the design graphically represented in Figure 10. In the figure, R, resp., $\widehat{R}$, denotes the `AES` round function, resp., the `AES` final round function, without the `AddRoundKey` operation. In other words, R consists of `SubBytes`, `ShiftRows`, and `MixColumns`, in this order, whereas $\widehat{R}$ consists of `SubBytes` and `ShiftRows`. The symbol $\circledS$ denotes the splitting of a state into s slices, and $\circledcirc$ the concatenation of s slices into a single $128\,s$-bit value. If $r_{st} = 2$ and the number of rounds is odd, the last round of the last step of the cipher is a single last round omitting `MixColumns`, with no mixing layer between this round and the previous one. This case is included in the pseudocode given in Algorithm 1.

3.3 The Mixing Layer

For any permutation F, its action on a state $[c_i]$ is defined as $F([c_i]) = [c_{F(i)}]$.

For the 256-bit version of `Vistrutah`, the mixing layer is the same as in `ASURA`, i.e., it is the byte shuffle defined as follows

$$\zeta^{-1} = \left[\begin{array}{cccc|cccc|cccc|cccc} 0 & 2 & 4 & 6 & 8 & 10 & 12 & 14 & 16 & 18 & 20 & 22 & 24 & 26 & 28 & 30 \\ 1 & 3 & 5 & 7 & 9 & 11 & 13 & 15 & 17 & 19 & 21 & 23 & 25 & 27 & 29 & 31 \end{array}\right] . \tag{2}$$

This shuffle is implemented on NEON using the `VUZP` instruction. On AVX, we first apply the permutation $[0, 2, 4, 6, 8, 10, 12, 14, 1, 3, 5, 7, 9, 11, 13, 15]$ on both halves of the state, and then we swap cells 8-15 of the first half with cells 0-7 of the second half by using two two-source operand `shuffle` instructions.

For the 512-bit version of `Vistrutah` the mixing layer is the permutation

$$\zeta = \left[\begin{array}{cccc|cccc|cccc|cccc} 0 & 16 & 32 & 48 & 1 & 17 & 33 & 49 & 2 & 18 & 34 & 50 & 3 & 19 & 35 & 51 \\ 8 & 24 & 40 & 56 & 9 & 25 & 41 & 57 & 10 & 26 & 42 & 58 & 11 & 27 & 43 & 59 \\ 4 & 20 & 36 & 52 & 5 & 21 & 37 & 53 & 6 & 22 & 38 & 54 & 7 & 23 & 39 & 55 \\ 12 & 28 & 44 & 60 & 13 & 29 & 45 & 61 & 14 & 30 & 46 & 62 & 15 & 31 & 47 & 63 \end{array}\right] , \tag{3}$$

which is computed as described in (14).

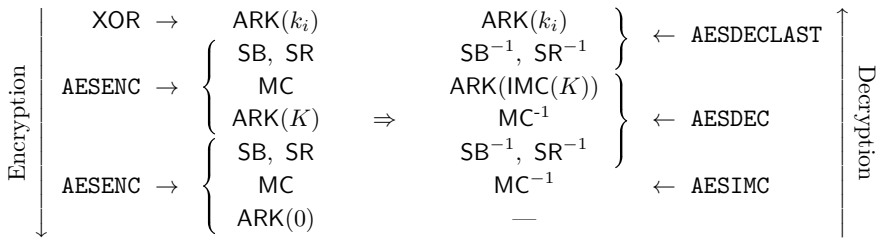


Figure 1: Step and Inverse Step of Vistrutah-R2 with the Intel AESNI instructions.

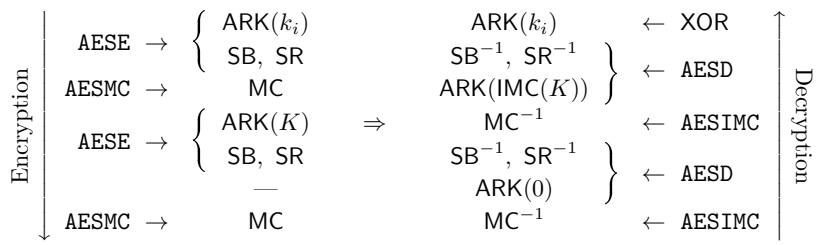


Figure 2: Step and Inverse Step of Vistrutah-R2 with the Arm NEON instructions.

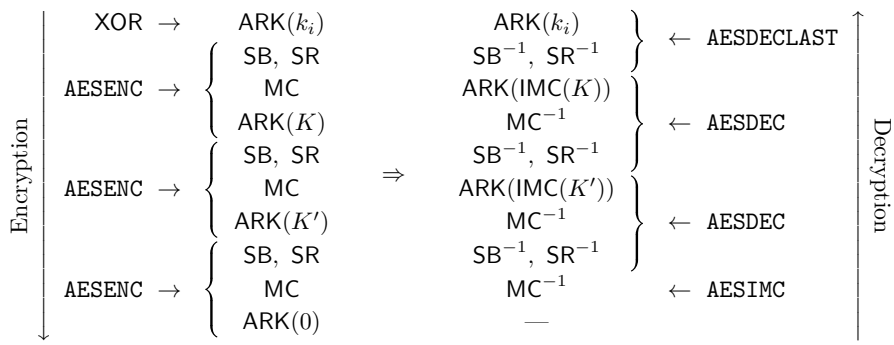


Figure 3: Step and Inverse Step of Vistrutah-R3 with the Intel AESNI instructions.

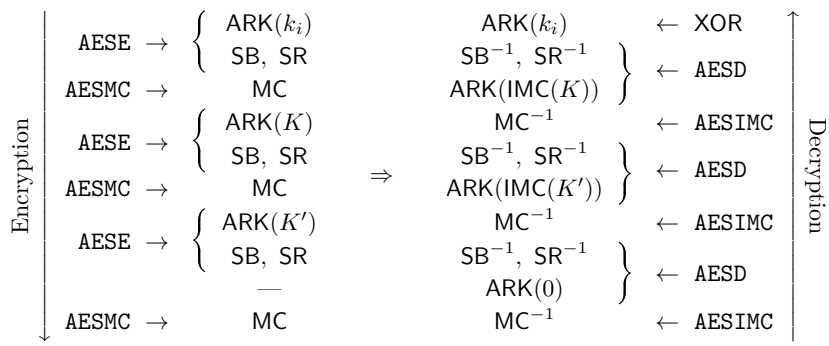


Figure 4: Step and Inverse Step of Vistrutah-R3 with the Arm NEON instructions.

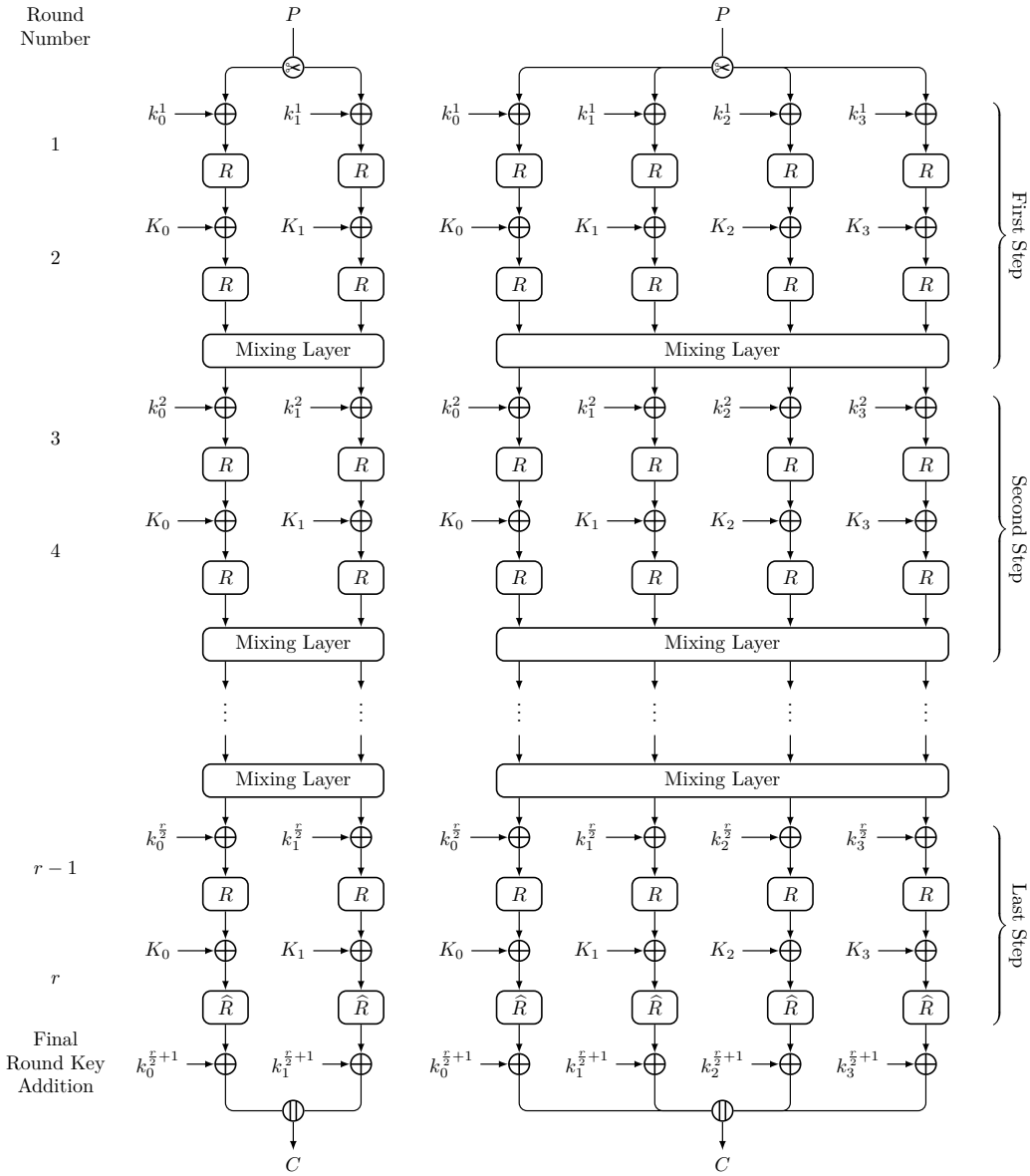


Figure 5: `Vistrutah`-256 and `Vistrutah`-512 (even number r of rounds).

The selected mixing layers effectively distribute state cells to ensure that all input bits influence every output bit non-linearly after one round following the initial mixing layer, and full diffusion is consistently achieved within three `AES` rounds.

3.4 The Key Schedule

For the 256-bit cells of each of the two ciphers, we split a 256-bit key into two 128-bit values as $K = K_0 \| K_1$, in other words $K_0 = K[0:127]$ and $K_1 = K[128:255]$. For the 512-bit cipher, a 512-bit *main key* is directly split into four 128-bit values as $K = K_0 \| K_1 \| K_2 \| K_3$.

For the 512-bit cipher with a 256-bit key, we apply a byte permutation to the 32 bytes

Algorithm 1 Pseudocode for Vistrutah Encryption.

Input: A plaintext block P (n bits long with $n \in \{256, 512\}$), and a key K (k bits long with $k \in \{256, 512\}$ and $k \leq n$).

Output: The ciphertext $\texttt{Vistrutah}_K(P)$

1: $X_0 \| \dots \| X_{s-1} \leftarrow P$ where $s = n/128$ ▷ *Split plaintext into slices*
2: $K_0 \| \dots \| K_{t-1} \leftarrow K$ where $t = k/128$ ▷ *Split key into slices*
3: **if** $(s = 4)$ **and** $(t = 2)$ **then**
4: $\quad K_2 \| K_3 \leftarrow \Psi((K_0 \| K_1))$ ▷ *Expand 256-bit key if state is 512-bit*
5: **for** $i = 0$ **to** $s/2 - 1$ **do** ▷ *Create first variable key*
6: $\quad k_{2i} \leftarrow K_{2i+1}$
7: $\quad k_{2i+1} \leftarrow K_{2i}$
8: **for** $j = 1$ **to** $\lfloor r/2 \rfloor$ **do**
9: $\quad$ **for** $i = 0$ **to** $s - 1$ **do**
10: $\quad\quad X_i \leftarrow R(X_i, k_i + \mathfrak{c}_j)$ ▷ *Two* AES *rounds*
11: $\quad\quad X_i \leftarrow R(X_i, K_i)$ **if** $j < r/2$ **else** $\widehat{R}(X_i, K_i)$
▷ *Note that $j < r/2$ is always true if r is odd*
12: $\quad$ **if** $j < r/2$ **then**
13: $\quad\quad$ **for** $i = 0$ **to** $s - 1$ **do** ▷ *Round-key update*
14: $\quad\quad\quad k_i \leftarrow \rho_{i \bmod 2}(k_i)$
15: $\quad$ **if** $j < \lfloor r/2 \rfloor$ **then**
16: $\quad\quad X_0 \| \dots \| X_{s-1} \leftarrow \texttt{MixingLayer}(X_0 \| \dots \| X_{s-1})$
17: **if** r is even **then**
18: $\quad$ **for** $i = 0$ **to** $s - 1$ **do** ▷ *Final round-key addition*
19: $\quad\quad X_i \leftarrow X_i + k_i$
20: **else** ▷ *This case needs another round*
21: $\quad$ **for** $i = 0$ **to** $s - 1$ **do**
22: $\quad\quad X_i \leftarrow \widehat{R}(X_i, k_i)$
23: $\quad\quad X_i \leftarrow X_i + K_i$
24: **return** $X_0 \| \dots \| X_{s-1}$

of the 256-bit key to obtain the second half of the 512-bit key. The chosen permutation is:

$$\Psi = [\ \begin{matrix} 30\ 29 & 8\ 23 & 10 & 9\ 20 & 3 & 22\ 21 & 0\ 31 & 2 & 1\ 28\ 11 \\ 14\ 13\ 24 & 7 & 26\ 25 & 4\ 19 & & 6 & 5\ 16\ 15 & 18\ 17\ 12\ 27 \end{matrix}\ ] \quad . \tag{4}$$

Ψ can be computed by two two-lane byte look-ups (on NEON these can be issued in parallel) or, alternatively, by first applying the following permutation of columns

$$\psi := [\ \ 0\ \ 5\ \ 2\ \ 7\ \ \ \ 4\ \ 1\ \ 6\ \ 3\ ] \tag{5}$$

and then applying the permutation

$$\phi = [\ 14\ 13\ \ 8\ \ 7\ \ \ 10\ \ 9\ \ 4\ \ 3\ \ \ \ 6\ \ 5\ \ 0\ 15\ \ \ \ 2\ \ 1\ 12\ 11\ ] \tag{6}$$

to the cells of each of the two slices of the result (this is a better approach for AVX, where ψ is computed by two long-word-wise `shuffle` operations issued in parallel).

We then adopt an alternating-key approach where two types of round key values are added in an alternating manner: one is a fixed value for all steps, and the other one is updated using in-slice permutations. (This bears similarities with the tweak schedule of QARMAv2 [ABD+23], but in our case only one round key is permuted.) Let ρ_0, ρ_1 be two suitable permutations on 16 elements. We define the key schedule for the 256-, resp., 512-bit version as follows:

$$k^1 = [\, k_0^1,\, k_1^1 \,] \leftarrow [\, K_1,\, K_0 \,]$$
$$k^{r+1} = [\, k_0^{r+1},\, k_1^{r+1} \,] \leftarrow [\, \rho_0(k_0^r),\, \rho_1(k_1^r) \,] \quad \text{for } r \geq 1 \ ,$$

where k^i is the variable key used in step number i, resp.,

$$k^1 = [\, k_0^1,\, k_1^1,\, k_2^1,\, k_3^1 \,] \leftarrow [\, K_1,\, K_0,\, K_3,\, K_2 \,]$$
$$k^{r+1} = [\, k_0^{r+1},\, k_1^{r+1},\, k_2^{r+1},\, k_3^{r+1} \,] \leftarrow [\, \rho_0(k_0^r),\, \rho_1(k_1^r),\, \rho_0(k_2^r),\, \rho_1(k_3^r) \,] \quad \text{for } r \geq 1 \ .$$

For the 256-bit version ($s = 2$) the permutations are

$$\begin{cases} \rho_0 = p_4 := [\ 9\ \ 7\ 13\ 14\ \ \ 0\ 10\ \ 3\ \ 5\ \ \ 1\ \ 2\ 15\ \ 4\ \ \ 6\ 12\ 11\ \ 8\ ] & (7) \\ \rho_1 = p_5 := [\ 12\ \ 8\ \ 1\ \ 9\ \ 15\ \ 4\ \ 0\ \ 3\ \ 14\ 10\ \ 6\ \ 7\ \ 2\ \ 5\ 13\ 11\ ] \ , & (8) \end{cases}$$

which are taken from [BDF24]. For the 512-bit version ($s = 4$) we use

$$\begin{cases} \rho_0 = \sigma_5 \ := [\ \ 5\ \ 6\ \ 7\ \ 8\ \ \ 9\ 10\ 11\ 12\ \ 13\ 14\ 15\ \ 0\ \ \ 1\ \ 2\ \ 3\ \ 4\ ] & (9) \\ \rho_1 = \sigma_{10} := [\ 10\ 11\ 12\ 13\ \ 14\ 15\ \ 0\ \ 1\ \ \ 2\ \ 3\ \ 4\ \ 5\ \ \ 6\ \ 7\ \ 8\ \ 9\ ] \ , & (10) \end{cases}$$

i.e. circular shifts of a slice, as a vector of cells, by 5 and 10 places to the left.

The rationale for this key schedule comes from both performance and security arguments. Key-derived values are added in two places during each step. In the first place, when a value v is added during encryption, its inverse `MixColumns` transform $\mathsf{MC}^{-1}(v)$ is added during decryption. In the second place, identical key-derived values are used for both encryption and decryption. To optimize performance, we use a fixed round key transformed by inverse `MixColumns` in the first place. In the second place, we use iteratively permuted values. This approach minimizes the number of inverse `MixColumns` operations needed during decryption. Therefore, we need a single inverse `MixColumns` transformation to adapt the key schedule to decryption. Besides the obvious performance benefit, any relation between key cells that prove useful for an attack will hold in one step but not (necessarily) in the other steps, thereby reducing the probability of successful attacks.

3.5 The Round Constants

The round constants are added to the state along with the variable round key, once per step. The cipher's structure ensures they affect every cell of the state by the end of the first round following the successive mixing layer. We have tried to exploit the fact that the round constants are added only to one slice of the state for the 512-bit variant, however so far without success. This said, the performance of `Vistrutah`-512 degrades only by 2 to 5 % due to the additional load and addition of the round constants for the second slice.

Following what is now common practice [BCG+12, KLMR16, BLLS22], the round constants are taken from the base-16 representation of the fractional part of π, cf. Figure 6. The first and last constants are set to zero. While we explored programmatic generation of the constants, loading precomputed constants from memory proved more efficient due to prefetching and caching.

Remark 1. For a round-based HW implementation, keeping a small *Real-Only Memory* (ROM) to access the round constants can be expensive in terms of area and also penalize performance. A better approach consists in using a *Linear Feedback Shift Register* (LFSR) to generate the round constants on-the-fly from an initial value. For decryption, the inverse LFSR can be used to generate the round constants in reverse order from the last value. This approach is described in Appendix C. We note that our entire cryptanalysis does not make use of the values of the round constants, and therefore holds also for the programmatically generated set.

$$
\begin{aligned}
\mathfrak{c}_1 &= [\,24\,3F\,6A\,88 \quad 85\,A3\,08\,D3 \quad 13\,19\,8A\,2E \quad 03\,70\,73\,44\,] \\
\mathfrak{c}_2 &= [\,A4\,09\,38\,22 \quad 29\,9F\,31\,D0 \quad 08\,2E\,FA\,98 \quad EC\,4E\,6C\,89\,] \\
\mathfrak{c}_3 &= [\,45\,28\,21\,E6 \quad 38\,D0\,13\,77 \quad BE\,54\,66\,CF \quad 34\,E9\,0C\,6C\,] \\
\mathfrak{c}_4 &= [\,C0\,AC\,29\,B7 \quad C9\,7C\,50\,DD \quad 3F\,84\,D5\,B5 \quad B5\,47\,09\,17\,] \\
\mathfrak{c}_5 &= [\,92\,16\,D5\,D9 \quad 89\,79\,FB\,1B \quad D1\,31\,0B\,A6 \quad 98\,DF\,B5\,AC\,] \\
\mathfrak{c}_6 &= [\,2F\,FD\,72\,DB \quad D0\,1A\,DF\,B7 \quad B8\,E1\,AF\,ED \quad 6A\,26\,7E\,96\,] \\
\mathfrak{c}_7 &= [\,BA\,7C\,90\,45 \quad F1\,2C\,7F\,99 \quad 24\,A1\,99\,47 \quad B3\,91\,6C\,F7\,] \\
\mathfrak{c}_8 &= [\,08\,01\,F2\,E2 \quad 85\,8E\,FC\,16 \quad 63\,69\,20\,D8 \quad 71\,57\,4E\,69\,] \\
\mathfrak{c}_9 &= [\,A4\,58\,FE\,A3 \quad F4\,93\,3D\,7E \quad 0D\,95\,74\,8F \quad 72\,8E\,B6\,58\,] \\
\mathfrak{c}_{10} &= [\,71\,8B\,CD\,58 \quad 82\,15\,4A\,EE \quad 7B\,54\,A4\,1D \quad C2\,5A\,59\,B5\,] \\
\mathfrak{c}_{11} &= [\,9C\,30\,D5\,39 \quad 2A\,F2\,60\,13 \quad C5\,D1\,B0\,23 \quad 28\,60\,85\,F0\,] \\
\mathfrak{c}_{12} &= [\,CA\,41\,79\,18 \quad B8\,DB\,38\,EF \quad 8E\,79\,DC\,B0 \quad 60\,3A\,18\,0E\,] \\
\mathfrak{c}_{13} &= [\,6C\,9E\,0E\,8B \quad B0\,1E\,8A\,3E \quad D7\,15\,77\,C1 \quad BD\,31\,4B\,27\,] \\
\mathfrak{c}_{14} &= [\,78\,AF\,2F\,DA \quad 55\,60\,5C\,60 \quad E6\,55\,25\,F3 \quad AA\,55\,AB\,94\,] \\
\mathfrak{c}_{15} &= [\,57\,48\,98\,62 \quad 63\,E8\,14\,40 \quad 55\,CA\,39\,6A \quad 2A\,AB\,10\,B6\,]
\end{aligned}
$$

Figure 6: The round constants used in `Vistrutah` (given as hexadecimal values).

4 Choice of the Components of `Vistrutah`

Selecting components for a cryptographic primitive is challenging since the relationship between design and computational cost is complex. Ideally, we would evaluate the *security per unit of computation time* for each design variant by combining exhaustive cryptanalysis with optimized implementations across multiple architectures. Since this is impractical, we model both analysis and performance evaluation at a manageable level.

Our approach has two key steps. First, we develop a simplified model of the *critical path* of the cipher in software. Second, we calculate the minimum number of active cells in differential characteristics (including RK ones) at various round counts. This allows us to identify designs that meet security thresholds while minimizing computational cost. To choose between variants with similar costs, we also consider concrete implementations as well as secondary factors like register usage to make the final design decisions.

The following two subsections detail our models for evaluating performance and security.

4.1 A Model of the Computational Cost of a Cipher Design

Performance comparisons of cryptographic algorithms in software are unavoidably tied to specific architectures, microarchitectures, and toolchains. This makes it difficult to transitively compare algorithms — a comparison of Algorithm A to Algorithm B in one paper and one of Algorithm B to Algorithm C in a second paper do not reliably translate to a comparison of Algorithm A to Algorithm C.

For the types of cipher that we consider, this task is simplified by the fact that we only use a few types of operations with a well understood behavior, i.e. those listed in Section 2.

We assume a load-store architecture with at least 16 128-bit vector registers. We also always use the *same* `AES` operation in parallel on all slices of the cipher. The reason is that we need to model a minimum common denominator: Arm NEON implementations can execute up to four independent `AES` instructions per cycle, while other architectures, such as Arm SVE2 and intel AVX512, optimize SIMD operations across wider vectors. Even though we do not benchmark implementations optimized for SVE2 or AVX512 in this paper, we expect the latter to be widely available in the near future. Our designs should not prevent future implementations from taking advantage of these features.

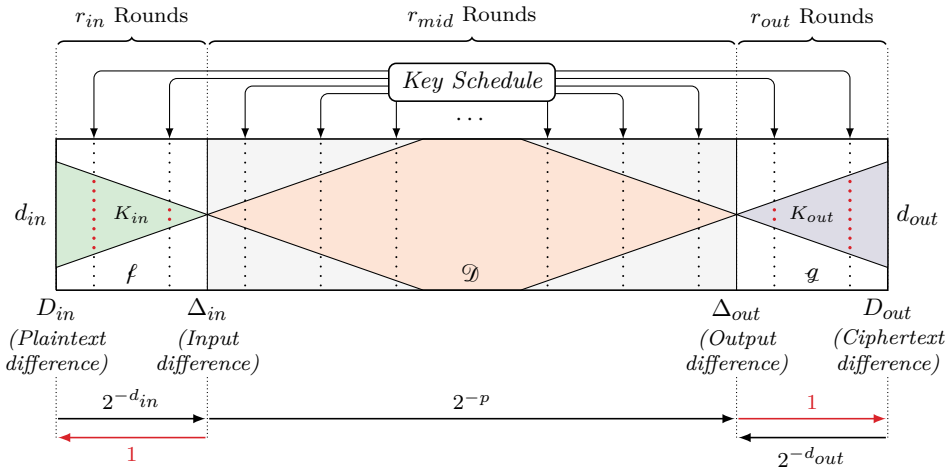


Figure 7: Key Recovery.

The cost of a step function is modeled as the number of groups of operations that can execute in parallel, accounting for their dependencies. We limit each group to a maximum of four operations. This level of parallelism is common in modern processors. On Arm, we assume the microarchitecture fuses instruction pairs `AESE`/`AESMC` and `AESD`/`AESIMC`, provided they are close to each other in the code, and we count them as single instructions. While this is obviously a rough estimate, we find that it is strongly correlated with the actual performance of the cipher variants we have implemented and benchmarked.

4.2 The Role of a Preliminary Differential and Linear Cryptanalysis

For `AES`-like ciphers, resistance against differential and linear cryptanalysis is typically assessed by finding the minimum number of rounds needed to achieve a given security level. A security level of p bits means that the probability of any useful *differential characteristic* [BS91] is at most 2^{-p}.

This assessment involves determining the minimum number of active S-boxes in any differential characteristic across a portion of the cipher, assuming the transform is an iterated Markov cipher. If the S-box S has a maximum differential probability $p_{max}(S)$, then multiplying this value by the number of active S-boxes yields an upper bound on the probability of differential characteristics. For the `AES` S-box, $p_{\max}(S) = 2^{-6}$.

A differential characteristic with probability 2^{-p} can be extended by adding rounds at the beginning and end. The number of rounds that can be added equals the number required for full diffusion, minus one — reaching the maximum number of rounds that can be possibly attacked using the characteristic. This is illustrated in Figure 7. Full diffusion must also be achieved from both directions within the distinguisher to strengthen against impossible differential (ID) [BBS99a] and miss-in-the-middle [BBS99b] attacks.

Since attacks typically focus on the input and output differences of a characteristic, i.e., a *differential*, rather than the characteristic itself, the actual transition probability can be significantly higher than that of the best characteristic. This means the actual security level may be lower than expected. Following common practice, we add a security margin by either increasing the security level by 20 % or increasing the number of rounds needed to reach the base security level by 20 %, whichever is greater.

There are several methods to establish lower bounds on the number of active cells in a characteristic. These range from ad-hoc arguments to computational approaches using brute force or custom tools [Mat93, FJP13, AA20], to the use of solvers for *Mixed-integer Linear*

Programming (MILP) [MWGP11, AC11, WW11, SHW+14, XZBL16, YML+17, FWG+16, ZZ18, WWHL18, YTC20, BA20, XZH20, QDW+21, IS22], *Boolean SATisfiability problem* (SAT)/*Satisfiability Modulo Theories* (SMT) [CB07, RS09, KY10, MP13, AJN14, KLT15, QCW16, SHY16, EKKT18, ZS20, SWW21, LW22, SII23, CLH+23, CXTQ23], and more general Constraint Programming [GMS16, SGL+17, SSD+18, ENP19, DDH+20, HSE23, DLP23, CHNE24, HGSE24].

We find the characteristics by the MILP approach, which is well suited to cell-wise analysis. In Tables 2 and 3 we report at which numbers of rounds, including the diffusion rounds, target security levels are reached. This data is derived from Tables 9 to 15, which show the minimal active cell counts across several cipher variants.

Regarding linear cryptanalysis, [KLW17] shows that linear key schedules do not significantly strengthen attacks ("*Choosing any linear key schedule and random round constants is on average as good as having independent round keys.*") and that tweak schedules do not introduce new linear characteristics. This allows us to simplify our analysis in the sense that a single-key analysis suffices [BJK+16], unlike in differential cryptanalysis. Since our step components are just the AES round function and either cell permutations or *Almost Maximal-Distance Separable* (AMDS)/*Maximum-distance Separable* (MDS) matrices, our differential cryptanalysis results directly apply to linear cryptanalysis as well.

Remark 2. Some cell counts in the RK model warrant a comment. Some reported values exceed those of the non-RK attacks — which may look absurd at first glance. The reason is that we enforce a non-zero key difference in the models, i.e., we describe *pure* RK differential attacks, which do not degenerate into non-RK, i.e., single-key differential attacks. If a cell count for the RK model is not smaller than that for the non-RK model, then the RK differential attacks are not more powerful than single-key ones. This is an indicator for both the quality of the key schedule and overall design robustness. This also gives an indication of the security level achievable in a potential extension of Vistrutah to a TBC following the Tweakey approach [JNP14]. We recall that Tweakey consists in adding a value derived from the tweak to the state at each round which is computed using the same permutations as the round keys but, additionally, where all cells of the round tweak are multiplied by a non-zero value of GF(2^8) for each round.

4.3 Design Options and Choices for Vistrutah

4.3.1 Number of Rounds per Step

In [SSI22] the authors compared constructions with one and two AES rounds per step with constructions, and concluded that the latter offer better security. They also observe that 2R constructions are much faster, but a proper comparison at the same security level is missing. The authors of Pholkos [BLLS22] argue that two rounds per step is the natural choice. However, since most operations added to a step have costs comparable to an AES operation, we observe that increasing the number of rounds per step, and thus reducing the relative impact of the cost of the mixing layer may yield a faster design at the same security level even keeping into account the fact that the number of active cells per AES round is reduced. Ghidle [NSA+23] is such a 3R design.

For these reasons, we evaluated both 2R and 3R designs. The first design we consider for Vistrutah is a 2R design using a suitable cell shuffle for the mixing layer. We call this family of designs Vistrutah.A. Its step function is represented in Figures 1 and 2 for the Intel and Arm AES instructions, respectively, and Figure 10 represents of Vistrutah.A with an even number of rounds. Prompted by Ghidle, we also considered 3R constructions where the third round in a step is a "last" round. This sub-family is called Vistrutah.B.

Ghidle places two AESENCLAST/AESDECLAST instructions consecutively in each of its lanes, effectively forming single keyed S-boxes out of pairs of S-boxes separated only by a cell-shuffle layer and key addition. This means that some S-Boxes in successive rounds

cannot be considered independently but, instead, *as a single keyed S-Box*. Also, each lane in a single step of `Ghidle` can be viewed as a composition of a single normal `AES` round and one round of a modified `AES` round using this keyed S-box. Hence, using simple S-box counting instead of accounting for this observation, the designers may have overestimated `Ghidle`'s security level. Indeed, we have reproduced the bounds given by the designers of `Ghidle` by considering the two S-Boxes independently. It is easy to verify that the maximal transition probability of the keyed S-box varies between 2^{-5} and $2^{-3.5}$, depending on the key, which is much higher than the probability of 2^{-12} one would obtain by considering the two S-Boxes independently. This changes the differential probability estimation of `Ghidle`. For instance, consider a single step `Ghidle` construction. It is easy to see that a single active S-box in the second lane in the first round of `Ghidle` leads to one active S-box at the end of each of the first two rounds, and another four active S-boxes at the end of the step. Hence, by the naïve estimation, the differential probability of one step of `Ghidle` is at most 2^{-36}. When the first two rounds are viewed as a single modified `AES` round with a keyed S-Box, given any key, in a conservative sense, the maximal differential probability can be estimated at around $2^{-3.5-(4\cdot 6)} = 2^{-27.5}$. Similar arguments hold for estimating the security level of `Ghidle` under other cryptanalytic attacks. It turns out that, due to the presence of a keyed S-box in each step, a concrete cryptanalysis of `Ghidle` using our observation is non-trivial and beyond the scope of this paper.

`Vistrutah.B` remedies `Ghidle`'s potential issue by using `AESENCLAST/AESDECLAST` only as the final round of a step. To minimize the amount of inverse `MixColumns` operations on round keys, we use two fixed round keys: $K = K_0 \| K_1$ for the 256-bit cipher (or $K_0 \| K_1 \| K_0 \| K_1$ for the 512-bit version) in the first round, followed by $K' = K_1 \| K_0$ (or $K_1 \| K_0 \| K_1 \| K_0$) in the second round. A cell permutation is applied to the third round key at each step. Since this key is used only in the final round, no inverse `MixColumns` transformation is needed for decryption. The structure is shown in Figures 8, 9 and 10. To

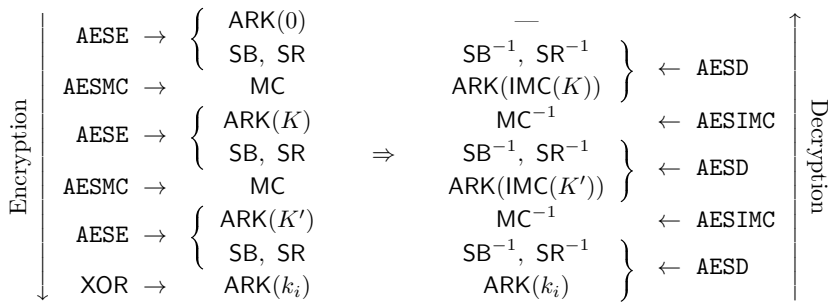


Figure 8: Step and Inverse Step of `Vistrutah.B` with non-zero keys (using NEON).

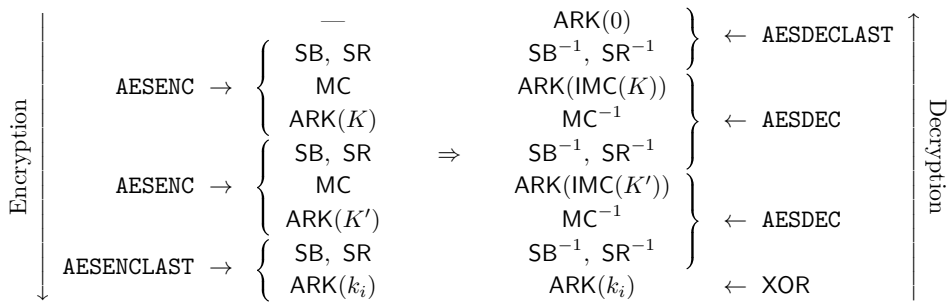


Figure 9: Step and Inverse Step of `Vistrutah.B` with non-zero keys (using AESNI).

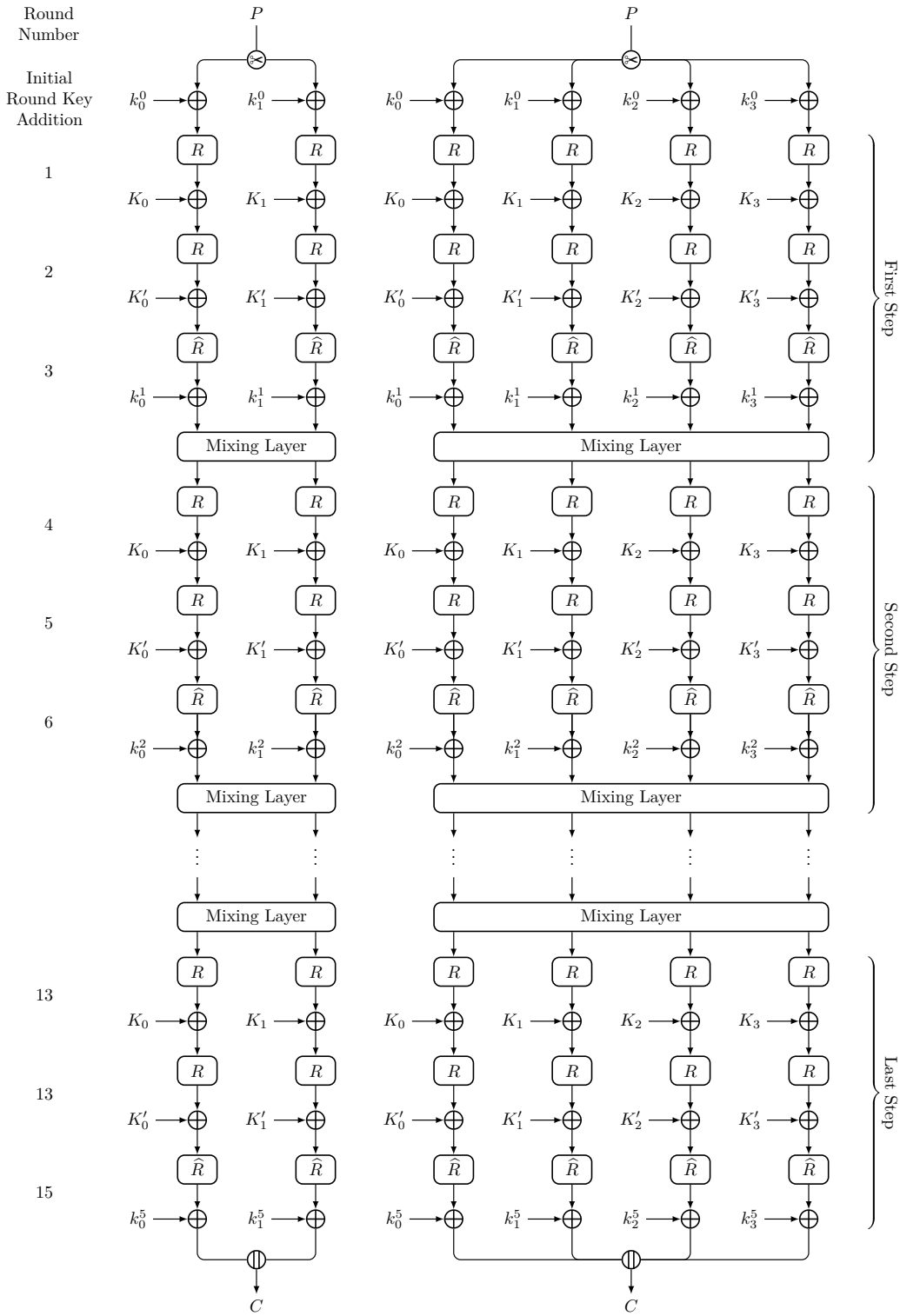


Figure 10: Vistrutah.B-256 and Vistrutah.B-512 $(3 \mid r)$.

avoid consecutive S-Box applications without a linear mixing with other cells in between, we cannot use a permutation as the mixing layer. We consider two alternatives. The first is to use an AMDS matrix, which is efficient but provides slow diffusion. The second is to apply an MDS matrix across the state's tubes by means of an (inverse) `MixColumns` operation — to ensure full diffusion at the beginning of the following step. In the case $s = 4$, this would require: (i) transposing state columns with tubes (for instance, using ζ as in (15), which, while not strictly a transpose, achieves the desired effect); (ii) applying the (inverse) `MixColumns`; and (iii) transposing the state back. The case $s = 2$ is similar (by considering the upper and lower half-vectors of each slice separately). This approach is expensive, but we still evaluate the corresponding security and implement such a mixing layer in order to verify the effectiveness of our latency model.

4.3.2 Choosing a Mixing Layer

We consider two types of mixing layer: cell shuffles and the application of a circulant binary AMDS matrix (with first row $[0, 1, 1, 1]$) to the state. For $s = 4$ designs, the matrix operates on the four slices, while for $s = 2$ it applies to four half-slices, where each of the two slices of the state is divided into two halves (Cells 0–7 and 8–15).

In all variants of the `Vistrutah` *family we have studied, the highest cell counts are given by certain cell shuffles.* We evaluated numerous shuffle options for the mixing layer, including column-rigid permutations, combinations with random in-slice permutations, rotations of the state cube around diagonals, and other exotic variants that could be implemented efficiently. Our analysis revealed that shuffles distributing cells from each slice across all columns performed best. This led us to focus on permutations with these properties that could be realized with minimal operationsideally just one or two operation groups. Using the methods from Subsection 4.2 to bound active cell counts, we found that matrix-based mixing layers consistently underperformed compared to cell shuffles.

In `Vistrutah.A`, we achieve full diffusion in just three rounds using a cell shuffle if the latter distributes the cells of the input slice to it in such a way that after the next `ShiftRows` operation, each of the columns of the state contains a cell from the input slice. (Cf. Subsection 6.1 for more precise statements.)

Let us consider a few variants of `Vistrutah.B` rounds. Using three full rounds and a shuffle mixing layer in a step results in weaker diffusion compared to other approaches. If the third round is a last round, and the mixing layer is an AMDS matrix, each slice only affects three other slices in the next step, requiring two mixing layers for full diffusion. While MDS matrices achieve faster full diffusion (4 rounds) and provide 25 % higher active cell counts in single-key scenarios, the steps get 50 % more expensive. Both the active cell counts shown in Tables 11 and 15 and our implementations demonstrate that using MDS matrices in `Vistrutah.B` is inefficient.

Remark 3. For `Vistrutah.B`-512, we found that a key schedule applying a circular rotation by 9 cells to all four slices yields some of the best cell counts. Any key schedule that applies the same permutation to all four slices commutes with MDS and AMDS mixing layers. Then, we can bring the round key addition across the mixing layer and substitute zero keys in `AES` instructions with the matrix-transformed round key. This reduces the cost of encryption by one operation group on NEON and similarly improves decryption by one operation group on AVX. In the timings of encryption of `Vistrutah.B`-512 on Apple Silicon (Table 16) and of decryption on the Intel i7 CPU (Table 17) we give both the timings for the implementation with this optimization and, in parentheses, without it.

4.3.3 On the Spaces of State and Key Shuffles

In order to find good shuffles for the mixing layer and the key schedules, we first considered various classes of permutations, as well as many random ones.

In light of Subsection 2.2, we prioritize permutations which are built-in in the most common ISAs, or simple combinations of such built-ins.

This is made necessary by two reasons. First, we want to make the state shuffles as fast as possible and reduce register usage. Therefore, generic parametrized permutation instructions, especially cross-lane ones, are used only as a last resort. Second, most papers that test thousands of (usually randomly chosen) permutations perform MILP or CP evaluations using very few rounds, for instance, up to 6 or 8 rounds. The reason is that the solving time grows exponentially with the number of variables, hence of rounds, and problems over, say, 12 rounds, can easily take days each to solve. This is particularly true for RK models. However, running many tests over more rounds, we notice that in many cases the behavior can stabilize around a different pattern than the initial one: in particular, some permutations that are very good on very few rounds (i.e., many active cells) then become ultimately not more efficient than others, or even less efficient.

Therefore, we chose to test fewer permutations and over more rounds, excluding early only those that perform very poorly on the first few rounds, The tables we prepared represent (a cross-section of an analysis that took) several years of CPU time. These considerations apply to both state shuffles and in-slice permutations for the key scheduling.

4.3.4 Options for the State Shuffles

After the initial experimentation with both structured and random permutations, patterns emerged which seemed to predict the effectiveness of a shuffle, and which can be used to restrict the search space.

If the cells from any fixed column are transported to a single column — either rigidly or with some in-column permutation — we do not seem able to get higher counts of cells than the counts displayed in Tables 9, 10, 12 and 13 for Pholkos' shuffles Φ for both $s = 2$ and $s = 4$, cf. (19) and (11).[1] However, we can do better than these shuffles, including Φ, with regard to the number of active cells in RK models. Note that these models also need a key schedule to be defined, cf. Subsubsection 4.3.7. For instance, if the state shuffle sends the sixteen cells of a slice to sixteen different columns, which is also needed maximize diffusion, a higher number of active cells could be observed in RK models.

Remark 4. Entries marked with upper bounds in Table 13 represent interrupted MILP solver runs. We stopped these runs early when the results became uninteresting. This happened when upper bounds were too low or when the results would not affect our component choices. For example, we stopped the solver runs for 10- and 12-round `Vistrutah.A-`Φ–(ξ, p_4, p_5) since it clearly could not significantly outperform `Vistrutah.A-`Φ–$(\xi, \sigma_5, \sigma_9)$, whereas the simple key schedule shifts are preferable to permutations p_4 and p_5, as the latter need to block two registers for lookup tables.

We start with the problem of constructing suitable shuffles for $s = 4$, since it is the more interesting one, and then we consider the case $s = 2$. We only present a few of the numerous shuffles we have considered, as the vast majority of them could be soon discarded due to their high computational cost or low active cell counts.

4.3.5 State Shuffles for $s = 4$

The `Pholkos` shuffle Φ applies the following permutation to the columns of the state

$$\Phi = [\ \ 0\ \ 5\ 10\ 15\ \ \ \ 4\ \ 9\ 14\ \ 3\ \ \ \ 8\ 13\ \ 2\ \ 7\ \ \ 12\ \ 1\ \ 6\ 11\ \ ]\ . \tag{11}$$

Implementing Φ requires three groups of operations on AVX and four on NEON.

[1] In the tables, the notation (F, π_0, π_2), F denotes a state permutation or bijective function, while π_0 and π_1 are the byte shuffles for even and odd numbered slices.

The following *transposition* of the columns of the state

$$\nu = [\quad 0\ \ 4\ \ 8\ 12 \quad\quad 1\ \ 5\ \ 9\ 13 \quad\quad 2\ \ 6\ 10\ 14 \quad\quad 3\ \ 7\ 11\ 15\ \ ] \tag{12}$$

usually gives the same active cell counts as for Φ but requires just two groups of operations on both architectures. Let A, B, C, D be four 128-bit vectors representing a state. Then, ν is implemented as:

$$\nu\ :\ \begin{cases}(A,C) = \texttt{TRN.64}(A,C)\\(B,D) = \texttt{TRN.64}(B,D)\end{cases} \quad \text{and} \quad \begin{cases}(A,B) = \texttt{TRN.32}(A,B)\\(C,D) = \texttt{TRN.32}(C,D)\end{cases},$$

on NEON, where a `TRN` operation is actually a `TRN1`/`TRN2` instruction pair applied in parallel to the two arguments. In our notation, the operations in each braced group are performed simultaneously and the destination registers are updated only after the entire group has been executed. On AVX we have found the following sequence of operations

$$\nu\ :\ \begin{cases}(A,B) = \texttt{unpack.32}(A,B)\\(C,D) = \texttt{unpack.32}(C,D)\end{cases} \quad \text{and} \quad \begin{cases}(A,B) = \texttt{unpack.64}(A,C)\\(C,D) = \texttt{unpack.64}(B,D)\end{cases}, \tag{13}$$

where `unpack.N` is a pair of `unpacklo.N`/`unpackhi.N` instructions.

We now present some NEON `ZIP`/`UZP`-based permutations. We define

$$\zeta\ :\ \begin{cases}(A,B) \leftarrow \texttt{ZIP.8}(A,B)\\(C,D) \leftarrow \texttt{ZIP.8}(C,D)\end{cases} \quad \text{and} \quad \begin{cases}(A,C) \leftarrow \texttt{ZIP.16}(A,C)\\(B,D) \leftarrow \texttt{ZIP.16}(B,D)\end{cases}, \tag{14}$$

where a `ZIP` operation is a `ZIP1`/`ZIP2` instruction pair — one step in the above formula is thus a group of four operations. On Intel AVX the instructions corresponding to `ZIP1` and `ZIP2` are `unpacklo` and `unpackhi`. Therefore, for both architectures, two groups of operations are required. The resulting permutation of cells is

$$\begin{aligned}\zeta = [\quad & 0\ 16\ 32\ 48 \quad 1\ 17\ 33\ 49 \quad 2\ 18\ 34\ 50 \quad 3\ 19\ 35\ 51\\ & 8\ 24\ 40\ 56 \quad 9\ 25\ 41\ 57 \quad 10\ 26\ 42\ 58 \quad 11\ 27\ 43\ 59\\ & 4\ 20\ 36\ 52 \quad 5\ 21\ 37\ 53 \quad 6\ 22\ 38\ 54 \quad 7\ 23\ 39\ 55\\ & 12\ 28\ 44\ 60 \quad 13\ 29\ 45\ 61 \quad 14\ 30\ 46\ 62 \quad 15\ 31\ 47\ 63\ \ ]\ .\end{aligned} \tag{15}$$

(Note that, with respect to a more "elegant" shuffle, which consists of a transposition of the tubes with the columns, the second and third slices are swapped. This does not impact the results, but uses fewer temporary registers.) On NEON it is easy to compute ζ^{-1}, the inverse of ζ, since the architecture provides the inverse of `ZIP1`/`ZIP2` in the form of the `UZP1`/`UZP2` instruction pair. In other words:

$$\zeta^{-1}\ :\ \begin{cases}A, C \leftarrow \texttt{UZP.16}(A,C)\\B, D \leftarrow \texttt{UZP.16}(B,D)\end{cases} \quad \text{and then} \quad \begin{cases}A, B \leftarrow \texttt{UZP.8}(A,B)\\C, D \leftarrow \texttt{UZP.8}(C,D)\end{cases}$$

However, AVX does not provide the inverses of `unpacklo`/`unpackhi`, and a longer sequence of operations must be used. We determine these operations by first looking at ζ^{-1} as a cell permutation:

$$\begin{aligned}\zeta^{-1} = [\quad & 0\ \ 4\ \ 8\ 12 \quad 32\ 36\ 40\ 44 \quad 16\ 20\ 24\ 28 \quad 48\ 52\ 56\ 60\\ & 1\ \ 5\ \ 9\ 13 \quad 33\ 37\ 41\ 45 \quad 17\ 21\ 25\ 29 \quad 49\ 53\ 57\ 61\\ & 2\ \ 6\ 10\ 14 \quad 34\ 38\ 42\ 46 \quad 18\ 22\ 26\ 30 \quad 50\ 54\ 58\ 62\\ & 3\ \ 7\ 11\ 15 \quad 35\ 39\ 43\ 47 \quad 19\ 23\ 27\ 31 \quad 51\ 55\ 59\ 63\ \ ]\ .\end{aligned} \tag{16}$$

This suggests to first regroup the bytes in each vector (slice) into the 32-bit words of the destination operation — that is, to form the groups $[0, 4, 8, 12]$, $[1, 5, 9, 13]$, $[2, 6, 10, 14]$,

and $[3, 7, 11, 15]$ out of the first vector, and so on. This can be done by transposing the bytes in a vector viewed as a 4×4 matrix of bytes. This permutation is described by (12), but operating on bytes, so we call it ν_8. After applying ν_8 to all four slices, we need just to transpose the resulting 4×4 matrix of 32-bit words, i.e. apply ν at the long word level. This requires an additional group of operations w.r.t. the NEON version, i.e.

$$\zeta^{-1} \;:\; \begin{cases} A, B = \texttt{unpack.32}(\nu_8(A), \nu_8(B)) \\ C, D = \texttt{unpack.32}(\nu_8(C), \nu_8(D)) \end{cases} \quad \text{and} \quad \begin{cases} A, B = \texttt{unpack.64}(A, C) \\ C, D = \texttt{unpack.64}(B, D) \end{cases} .$$

A variation of ζ^{-1}, obtained by applying a circular rotation by one cell to each slice of (16), reaches the highest number of active cells in many cases, including for $r = 6$, 8:

$$\xi = \left[\begin{array}{cccc|cccc|cccc|cccc} 60 & 0 & 4 & 8 & 12 & 32 & 36 & 40 & 44 & 16 & 20 & 24 & 28 & 48 & 52 & 56 \\ 61 & 1 & 5 & 9 & 13 & 33 & 37 & 41 & 45 & 17 & 21 & 25 & 29 & 49 & 53 & 57 \\ 62 & 2 & 6 & 10 & 14 & 34 & 38 & 42 & 46 & 18 & 22 & 26 & 30 & 50 & 54 & 58 \\ 63 & 3 & 7 & 11 & 15 & 35 & 39 & 43 & 47 & 19 & 23 & 27 & 31 & 51 & 55 & 59 \end{array} \right] . \tag{17}$$

This shuffle requires one group of four additional operations to be computed, potentially making the step slower, while also reaching higher security levels in fewer rounds.

Also ζ, ζ^{-1} and ξ guarantee full diffusion after one step and a round.

The last option we consider consists of the following operations in order:

$$\begin{cases} A, B = \texttt{TRN.32}(A, B) \\ C, D = \texttt{TRN.32}(C, D) \end{cases}, \quad \begin{cases} A, C = \texttt{TRN.64}(A, C) \\ B, D = \texttt{TRN.64}(B, D) \end{cases} \quad \text{and} \quad \begin{cases} A, D = \texttt{TRN.16}(A, D) \\ B, C = \texttt{TRN.16}(B, C) \end{cases} .$$

This results in

$$\chi = \left[\begin{array}{cccc|cccc|cccc|cccc} 0 & 1 & 12 & 13 & 16 & 17 & 28 & 29 & 32 & 33 & 44 & 45 & 48 & 49 & 60 & 61 \\ 2 & 3 & 14 & 15 & 18 & 19 & 30 & 31 & 34 & 35 & 46 & 47 & 50 & 51 & 62 & 63 \\ 4 & 5 & 8 & 9 & 20 & 21 & 24 & 25 & 36 & 37 & 40 & 41 & 52 & 53 & 56 & 57 \\ 6 & 7 & 10 & 11 & 22 & 23 & 26 & 27 & 38 & 39 & 42 & 43 & 54 & 55 & 58 & 59 \end{array} \right] , \tag{18}$$

which also guarantees full diffusion in one step and one round.

4.3.6 State Shuffles for $s = 2$

The `Pholkos` shuffle Φ applies permutation

$$\Phi = [\; 0 \;\; 5 \;\; 2 \;\; 7 \;\;\; 1 \;\; 4 \;\; 3 \;\; 6 \;] \tag{19}$$

to the columns of the state. On AVX this is implemented by two independent `blend`s, and on NEON by using two (independent) two-register table-lookups, hence always in one group. A permutation with similar properties is

$$\nu = [\; 0 \;\; 4 \;\; 2 \;\; 6 \;\;\; 1 \;\; 5 \;\; 3 \;\; 7 \;]$$

which is easily implemented in NEON as two transpose instructions in a single operation group, and on AVX it needs two groups (each consisting of two `shuffle` instructions).

Other fast shuffles are based on the Arm NEON `ZIP`/`UZP` operations, such as, namely ζ that "zips" the two slices of a state as bytes

$$\zeta = \left[\begin{array}{cc|cc|cc|cc|cc|cc|cc|cc} 0 & 16 & 1 & 17 & 2 & 18 & 3 & 19 & 4 & 20 & 5 & 21 & 6 & 22 & 7 & 23 \\ 8 & 24 & 9 & 25 & 10 & 26 & 11 & 27 & 12 & 28 & 13 & 29 & 14 & 30 & 15 & 31 \end{array} \right] \tag{20}$$

and its inverse ζ^{-1}, i.e., Permutation (2), as well as the composition of ζ^{-1} with a circular shift of each slice,

$$\xi = \left[\begin{array}{cccc|cccc|cccc|cccc} 30 & 0 & 2 & 4 & 6 & 8 & 10 & 12 & 14 & 16 & 18 & 20 & 22 & 24 & 26 & 28 \\ 31 & 1 & 3 & 5 & 7 & 9 & 11 & 13 & 15 & 17 & 19 & 21 & 23 & 25 & 27 & 29 \end{array} \right] . \tag{21}$$

For $s = 2$, χ takes the form

$$\chi = \left[\begin{array}{cccc|cccc|cccc|cccc} 0 & 1 & 8 & 9 & 16 & 17 & 24 & 25 & 2 & 3 & 10 & 11 & 18 & 19 & 26 & 27 \\ 4 & 5 & 12 & 13 & 20 & 21 & 28 & 29 & 6 & 7 & 14 & 15 & 22 & 23 & 30 & 31 \end{array} \right] . \tag{22}$$

All these shuffles guarantee full diffusion in one step and one round.

4.3.7 Options for the Key Schedule

Choosing the right key schedule presents several fundamental challenges. We focus on the key design decisions that influenced our work.

The first decision is between linear and non-linear key schedules. We exclude the latter because they are computationally expensive, usually make encryption and decryption operations asymmetric in performance, and their security benefits are still unclear. Linear key schedules, on the other hand, may even provide stronger *proven* security guarantees (cf. Subsection 4.2). The main two categories are bit (or cell) permutations, and linear transformations such as matrix operations. Bit permutations, and especially cell shuffles, are straightforward to implement in MILP models and maintain exact estimates of the number of active round keys cells, leading to tighter security estimates. Invertible matrix operations combine key cell values, making it harder to restrict the activity of cells in RK differential characteristics. Based on these considerations, we base our key schedule on cell permutations. We note that there has been research on alternative key schedules for the `AES` to hinder RK cryptanalysis, for instance [DFJL18, KLPS17, BDF24], and these are also cell shuffles.

When selecting cell permutations, we evaluate their effect on RK differential cryptanalysis, as discussed in Subsection 1.1. We examine how the combination of state shuffle and key schedule shuffle affects cell dependencies: we can discard immediately many in-slice key shuffles by minimizing the number of cells that are added to cells affected by themselves through these shuffles. This is however, cumbersome, and the MILP solver reveals very low numbers of active cells for such combinations permutations at just 6 and 8 rounds.

Besides (7), (8), and all circular shifts of each slice, we evaluated many other permutations, among them ϕ from (6) and

$$\kappa := \left[\begin{array}{cccc|cccc|cccc|cccc} 11 & 15 & 3 & 7 & 12 & 0 & 4 & 8 & 1 & 5 & 9 & 13 & 2 & 6 & 10 & 14 \end{array} \right] , \tag{23}$$

$$\kappa^{-1} := \left[\begin{array}{cccc|cccc|cccc|cccc} 5 & 8 & 12 & 2 & 6 & 9 & 13 & 3 & 7 & 10 & 14 & 0 & 4 & 11 & 15 & 1 \end{array} \right] , \text{and} \tag{24}$$

$$\bar{\phi} := \left[\begin{array}{cccc|cccc|cccc|cccc} 11 & 12 & 1 & 2 & 15 & 0 & 5 & 6 & 3 & 4 & 9 & 10 & 7 & 8 & 13 & 14 \end{array} \right] . \tag{25}$$

Permutation (23) is taken from [KLPS17]. Permutation (25) is the composition of ϕ with a cell order reversal.

4.3.8 Options for the Key Stretching

For the 512-bit wide version of `Vistrutah`, we need to *stretch* the 256-bit key to 512 bits, in which case the influence of the key stretching method on the number of active cells is second only to the choice of Mixing Layer. We have considered various lightweight options:

- Apply a byte permutation to the 32 bytes of the 256-bit key to obtain the second half of the 512-bit key. We consider various permutations, including the identity.
- Follow the approach used in `Pholkos`, where the least significant ("leftmost") 256 bits of the expanded key are a copy of the *main key* K, and the most significant ("rightmost") 256 bits of the expanded key are computed by using a matrix multiplication with branch 4.

- Use an unkeyed AES round, in other words, compute $K_2 = \texttt{AESENC}(K_0, 0)$ and $K_3 = \texttt{AESENC}(K_1, 0)$. This, apart from the SBox application, is similar to the previous approach but with branch number 5. We also test the application of two unkeyed AES rounds, resulting in a function with branch number 8.

For all variants of Vistrutah *that we have analyzed, byte shuffling yields the best results among the four key stretching methods. While using an AMDS, resp., MDS matrix forces an active initial round key to have at least 4, resp., 5 active cells, it does not bring better results already after few rounds.* See in Table 13, Vistrutah.A-copy, Vistrutah.A-Φ, Vistrutah.A-1AES, resp., Vistrutah.A-2AES use the following key stretching functions: identity, the shuffle Ψ defined in (4), an unkeyed AES round, and two unkeyed AES rounds.

4.4 Costs of the Steps of Various Ciphers

In Table 1, we present the computational costs for encryption and decryption steps of Vistrutah, Pholkos, Ghidle, and ASURA based on our estimation methodology. We used Arm NEON operation grouping to count the groups of operations. The Intel AVX architecture shows at most a one-group difference in either encryption or decryption. Modern Intel architectures like AVX-512 or AVX-10, with their cross-lane shuffle capabilities, typically match NEON's group counts. The * notation indicates step functions requiring two-source permutation instructions. These may incur higher costs than single-source permutations, though surrounding instructions can sometimes mask this latency. We conservatively estimate four-source permutation instructions as requiring at least two groups, also marking them with * due to their potentially higher cost. While our implementation attempts to minimize this latency, only cycle-accurate simulation or actual implementation can precisely determine the step function's true cost. For a detailed breakdown of each cipher's step components, see Appendix D.

4.5 Final Selection of the Components

A combination of components must be chosen so that it reaches set security levels. So, first, we look at the results of active cell counting to determine how many rounds are needed to reach 256 and 512 bits of security by each evaluated combination, as well as the security levels increased by 20 %, i.e., 307 and 614 bits These are reported in Tables 2 and 3, where then the number of rounds is multiplied by the cost per round to obtain the cost to reach those security levels.

For instance, differential distinguishers for 512-bit Vistrutah.A-Φ–$(\zeta, \sigma_5, \sigma_{10})$ reach 512 bit security at 10 rounds in the single-key model and in the RK model with 256-bit keys. Thus, 14 rounds should not be attackable. Adding 20 % of rounds to the distinguisher, we see, with a minor rounding below, that 16 rounds should have sufficient security. If we include the RK model with 512-bit keys, we see that distinguishers over 12 rounds have a probability of at most $2^{-86.6} = 2^{-516} < 2^{-512}$. Adding 20 %, rounding to the closest integer, and 2+2 rounds, we obtain 18 rounds. The values for all security levels in Tables 2 and 3 do *not* contain the 20 % adjustment.

When comparing Vistrutah.B to 3R Vistrutah.A, three key factors emerge: First, for $s = 2$, Vistrutah.B's AMDS matrix-based mixing layer is more computationally expensive, negating its advantage of requiring one fewer operation in the encryption step. Therefore, Vistrutah.B does not seem interesting for $s = 2$. For $s = 4$, however, Vistrutah.B's encryption step is on par with 3R Vistrutah.A's and decryption is faster. This makes Vistrutah.B potentially interesting for $s = 4$.

Even before seeing concrete implementations, we can expect Ghidle and Vistrutah.B to perform similarly, with the latter possibly being slightly faster because the key schedule requires fewer instructions, even if the number of groups is the same. Therefore, in each

Table 1: Costs in numbers of instruction groups of various step functions for encryption and decryption. * denotes step functions with two source-permutation instructions, which may cause additional latency. † denotes decryptions that can require two or more additional groups of operations on AVX (instead of zero or one) than they do on NEON.

Cipher and Mixing Layer	Cost per step				Cost per AES round			
	256-bit ($s=2$)		512-bit ($s=4$)		256-bit ($s=2$)		512-bit ($s=4$)	
	Enc.	Dec.	Enc.	Dec.	Enc.	Dec.	Enc.	Dec.
Vistrutah.A–Φ	6*	7*	11*	11*	3	3.5	5.5	5.5
Vistrutah.A–$\nu/\zeta/\zeta^{-1}$	6	7	7	8	3	3.5	3.5	4
Vistrutah.A–ξ/χ	6*	7*	7*	8*	3	3.5	3.5	4
Vistrutah.A-3R-$\nu/\zeta/\zeta^{-1}$	7	8	8	9	2.33	2.67	2.67	3
Vistrutah.B–AMDS	8	8	8	8	2.67	2.67	2.67	2.67
Vistrutah.B–MDS	8	7†	11	10†	2.67	2.33	3.67	3.33
Pholkos (untweaked)	9*	9*	13*	15*	4.5	4.5	6.5	7.5
Ghidle	8	8	8	8	2.67	2.67	3	3
ASURA	7	8	—	—	3.5	4	—	—

Table 2: Costs in numbers of instruction groups at which various ciphers with $s = 2$ are estimated to reach set security levels against differential attacks where the adversary can control the input. The number of rounds is the minimum required to reach the corresponding probability, plus twice the number of rounds to achieve full diffusion, minus two. Pairs of cost values are for encryption and decryption, respectively. The row in boldface indicates the selected configuration. The marks * and † are defined as in Table 1.

Security level in bits:	256		307	
Rounds and Costs (E/D):	r	Cost	r	Cost
Vistrutah.A-2R-ζ^{-1}	**10**	**30/35**	**12**	**36/42**
Vistrutah.A-2R-$\Phi/\xi/\chi$	12	36*/42*	12	36*/42*
Vistrutah.A-2R-ν/ζ	12	36/42	12	36/42
Vistrutah.A-3R-ν	15	35/40	15	35/40
Vistrutah.B-AMDS	18	48	18	48
Vistrutah.B-MDS	12	32†/28†	15	40†/35†
Pholkos (untweaked)	12	54*	12	54*
Ghidle	18	48	18	48
ASURA	10	35	12	42

Table 3: Costs in numbers of instruction groups at which various ciphers with $s = 4$ are estimated to reach set security levels against differential attacks, similarly to Table 2.

Security level in bits:	256		307		512		614	
Rounds and Costs (E/D):	r	Cost	r	Cost	r	Cost	r	Cost
Vistrutah.A-Φ-2R-Φ	12	66*/78*	14	77*/91*	16	88*/104*	16	88*/104*
Vistrutah.A-Φ-2R-ζ^{-1}	12	42/48	12	42/48	16	56/64	20	70/80
Vistrutah.A-Φ-2R-ν/ζ	**12**	**42/48**	**14**	**49/56**	**14**	**49/56**	**18**	**63/72**
Vistrutah.A-Φ-2R-ξ/χ	12	48*/54*	12	48*/54*	14	56*/63*	16	64*/72*
Vistrutah.A-Φ-3R-ν	18	48/54	18	48/54	24	64/72	27	72/80
Vistrutah.B-AMDS	18	48	21	56	24	64	27	72
Vistrutah.B-MDS	12	44†/40†	12	44†/40†	18	66†/60†	18	66†/60†
Pholkos	10	65*/75*	12	78*/90*	14	91*/105*	14	91*/105*
Ghidle	18	54	21	63	24	72	27	81

case where a Vistrutah.A construction is preferred over Vistrutah.B, Vistrutah.A is then preferable over Ghidle as well, even with the latter's original security estimates.

When comparing designs that require the same number of rounds for full diffusion and achieve similar performance at comparable security levels, we prefer those that take more rounds as they provide better diffusion. For instance, in Table 3, the variants of Vistrutah.A using ν and ζ outperform the variant using ξ by this metric.

4.5.1 Selection of the Components for $s = 2$

If we only look at Table 2, all mixing layers are potential choices for Vistrutah.A with $r_{st} = 2$, and we can also consider Vistrutah.A with $r_{st} = 3$. Therefore, we need to consider the robustness of the key schedule, i.e., the cell counts for RK differential cryptanalysis per Table 10. We prioritize the cell counts for 4 to 10 rounds since, in concrete attacks, distinguishers over these numbers of rounds are extended to 10 to 16 rounds, and these are the round numbers we consider to define the long and short version of the ciphers. Longer distinguishers are not important. When matters are still close, we also take the timings of a concrete implementation into account. The choice of mixing layer falls on ζ^{-1} with p_4 and p_5 as the key scheduling permutations. Mixing layer ξ (also with p_4 and p_5) is a close second, and while encryption performance w.r.t. using ζ^{-1} is not worse, decryption performance is about 5-10 % worse on the Apple Silicon M3 computer, and about 25 % on our Intel machine — see Tables 16 and 17.

4.5.2 Selection of the Components for $s = 4$

For $s = 4$, we follow a similar approach to the $s = 2$ case. The most important cell counts are those for 6 to 12 rounds.

Let us first select the best option for Vistrutah.A. Based on Tables 1 and 3, we exclude Φ as a mixing layer due to its high cost. We do not immediately exclude ζ^{-1}, even though it has lower minimal numbers of active S-Boxes for cell-wise differential characteristics at higher round numbers (cf. Tables 12 and 13), because we need to consider the RK behavior. Beside ζ^{-1}, the other remaining candidates are ν, ζ, and ξ. While ν sometimes has lower active cell counts, both our operation group counts and implementations (cf. Tables 16 and 17) show that ν and ζ perform similarly, while ξ performs worse. Hence, ζ is our choice among these three candidates.

The choice between ζ with ζ^{-1} is not straightforward, but: (i) ζ has larger margins in the single-key model at the higher rounds count, which is important for the long version of the cipher. The margins for the short version are less significant, because, as we shall see, we limit the attacker's ability to control the key and the plaintext input. (ii) On Apple Silicon ζ and ζ^{-1} provide similar encryption performance, but on Intel, using ζ^{-1} encryption is 15-20 % slower than with ζ, even though decryption improves accordingly. As we prioritize the performance of encryption over decryption, we select ζ.

For the in-slice key scheduling permutations, using (23) with $s = 4$ would yield higher cell counts for pure RK differentials with 256-bit keys compared to our chosen permutations σ_5 and σ_{10}. However, σ_5 and σ_{10} perform better for pure RK differentials with 512-bit keys. In both cases the differences are small, and we choose σ_5 and σ_{10} because of the better behaviour with 512-bit keys, which is possibly more realistic in case an attacker can glitch the storage of the stretched key schedule.

For Vistrutah.B, the estimates in Table 3 sometimes approach those of Vistrutah.A. Concrete implementations (Tables 16 and 17) confirm its good projected performance. Using Table 3, we compare implementations with similar security levels: 10-round and 12-round Vistrutah.A versus 18-round Vistrutah.B, and 14-round Vistrutah.A versus 24-round and 27-round Vistrutah.B. While the performance differences are sometimes small, Vistrutah.A performs better overall. The need for short cipher versions further

favors `Vistrutah.A` due to its faster diffusion - 6 rounds of `Vistrutah.B` costs roughly the same as 4.5 rounds of `Vistrutah.A`, but `Vistrutah.A` achieves full diffusion in just 3 (always linearly, sometimes non-linearly) or 4 (non-linearly) rounds, depending on diffusion direction and starting point.

4.5.3 Post-Facto Remark on the Selection

Our evaluation of the cost of a step function per bit of security turns out to be a very good proxy for the actual cost. We selected the overall more inexpensive configurations. The cost values are also in agreement with the implementation results, which we detail next.

5 Implementing the Large Block Ciphers

5.1 Software Implementation

We implemented our ciphers in the C language (following the C11 standard) as both portable C code and optimized versions using NEON and AVX intrinsics.

For Arm architectures, we focused on NEON because SVE (Scalable Vector Extension) with 256-bit or longer vectors lacks widespread support, SVE with 128-bit vectors offers no significant performance advantage over NEON for this type of programs, and NEON already provides all the functionality we require.

For Intel architectures, we used AVX instructions, and only the `VPERMD` instruction from AVX2. We note that AVX2 with 256-bit vectors provides minimal performance gains over AVX for our use case. While AVX2 could theoretically parallelize some operations, it mainly serves to make the code more compact, and the underlying execution still processes the same number of 128-bit operations issued to the same number of ports.

We do not provide specific AVX512 implementations, since AVX512 tends to bring the same improvement to all programs of the types we consider, and it is also not yet widely deployed: many consumer intel CPUs use *performance* cores that support AVX512, but the feature is disabled when the *efficiency* cores on the same SoC do not implement it.

For `Vistrutah`, we identified step structures that achieve near-optimal performance on both Arm and Intel architectures for encryption, and minimize decryption overhead.

Our implementation uses identical keys for encryption and decryption. This design choice requires the decryption function to fast-forward the key schedule to match the final encryption round key. We implemented this efficiently through several approaches.

- For permutation-based key schedules, decryption starts by applying an initial precomputed permutation with minimal performance impact.
- For some state shuffles and key schedules, we can leverage mathematical properties:
  - If the mixing layer is an AMDS matrix, we use the parity of the step count to determine whether a matrix multiplication is required.
  - The `Pholkos` key-update permutations have order eight, i.e. their eighth powers are the identity permutation, and the fourth power of the `Pholkos`-256 key update is a simple in-slice byte permutation.
  - The key-update permutation of `Ghidle`-512 has order three.
- The `AES` and `Rijndael`-256-256 require special handling, since no fast-forwarding optimization is possible. We precompute the key schedules at operation start and explicitly erase them afterward. This is faster than inline computation, especially for decryption, since in the latter case we just consume the round keys in reverse.

Table 4: Performance comparisons of Vistrutah against other ciphers at the same security levels on an Apple Silicon M3 computer clocked at 3.67Ghz. The C compiler is Clang version 17.0. For Vistrutah we show the performance of both short and long versions. For AES-256 (resp., Rijndael-256-256), we process one, two, and four 128-bit blocks simultaneously under the same key (resp., one and two blocks). Both AES-256 and Rijndael-256-256 are timed with precomputed and with on-the-fly key schedule.

256-Bit Cipher	r	Time (ns)	
		Enc.	Dec.
Vistrutah	10	3.42	4.09
Vistrutah	14	4.39	6.29
Pholkos	16	7.71	9.77
Ghidle	24	9.52	10.07
ASURA	12	5.00	8.18
Simpira v2	15×2	5.62	5.19

512-Bit Cipher	r	Time (ns)	
		Enc.	Dec.
Vistrutah	12	7.69	10.25
Vistrutah	14	9.03	12.39
Vistrutah	18	11.78	16.66
Pholkos	20	34.18	39.12
Ghidle	24	14.28	14.77
Simpira v2	15×2	9.95	9.77

Versions with *precomputed* key expansion

Cipher	Blocks	r	Time (ns)	
			Enc.	Dec.
AES-256	1	14	1.53	1.53
AES-256	2	14	2.32	2.32
AES-256	4	14	3.97	3.91
Rijndael-256-256	1	14	7.81	7.69
Rijndael-256-256	2	14	13.18	13.18

Versions with *on-the-fly* key expansion

Cipher	Blocks	r	Time (ns)	
			Enc.	Dec.
AES-256	1	14	17.58	21.00
AES-256	2	14	18.98	23.99
AES-256	4	14	21.48	31.01
Rijndael-256-256	1	14	88.99	88.99
Rijndael-256-256	2	14	100.59	124.15

While this approach biases our comparisons towards the AES and Rijndael-256-256, we still provide better performance with our design. We also provide results for AES-128 and Rijndael-256-256 with a precomputed key schedule.

For all other ciphers, performance differences between encryption and decryption stem primarily from step structure variations rather than key schedule adjustments.

Finally, we do not manually unroll loops. Instead, we rely on modern CPU branch predictors to reduce the penalty associated with loops. We made exceptions only for AES-256 and Rijndael-256-256. For these ciphers, compilers would automatically unroll loops due to fixed round counts anyway. These exceptions maintain comparison fairness without biasing results in Vistrutah's favor.

5.1.1 Results

Table 4 shows timings for our NEON implementations of Vistrutah and its competition at comparable security levels. We also report timings on an Intel Core i7 13700K in Table 5. More complete sets of results are given in Tables 16 and 17 in the Appendix.

We report timings in nanoseconds rather than cycles per byte. The instruction counters on Apple Silicon CPUs do not follow the Arm specification for the optional *Data Watchpoint and Trace* (DWT) unit and are undocumented. Although the Apple instruction counter has been reverse engineered [Lem21, Lem23], the reported values may contain inconsistencies. We therefore chose to report timings in nanoseconds. The cryptographic operations execute quickly enough that context switches rarely occur during execution. As a result, the minimal timings strongly correlate with cycles per byte. To obtain *approximate* cycle-per-byte values, multiply the nanosecond timings by 3.67 for the Apple Silicon CPU and by 5 for the Intel i7 13700K, then divide by the block width in bytes. For example, this yields approximately 0.5 and 0.71 cycles per byte for Vistrutah-256 and Vistrutah-512,

Table 5: The same performance comparisons as in Table 4, on an Intel Core i7 13700K CPU clocked at 5 GHz. The C compiler is GCC version 11.0.4.

256-Bit Cipher	r	Time (ns)	
		Enc.	Dec.
`Vistrutah`	10	4.63	5.58
`Vistrutah`	14	7.35	8.65
`Pholkos`	16	10.63	15.06
`Ghidle`	24	11.13	11.85
`ASURA`	12	12.01	13.63
`Simpira` v2	15×2	3.89	3.89

512-Bit Cipher	r	Time (ns)	
		Enc.	Dec.
`Vistrutah`	12	11.82	17.08
`Vistrutah`	14	13.58	21.82
`Vistrutah`	18	19.38	29.90
`Pholkos`	20	29.82	47.40
`Ghidle`	24	19.72	21.03
`Simpira` v2	15×2	6.85	6.82

Versions with *precomputed* key expansion

Cipher	Blocks	r	Time (ns)	
			Enc.	Dec.
`AES-256`	1	14	1.55	3.12
`AES-256`	2	14	2.49	4.77
`AES-256`	4	14	5.04	7.25
`Rijndael-256-256`	1	14	12.41	14.09
`Rijndael-256-256`	2	14	22.72	24.00

Versions with *on-the-fly* key expansion

Cipher	Blocks	r	Time (ns)	
			Enc.	Dec.
`AES-256`	1	14	22.61	22.62
`AES-256`	2	14	34.80	35.33
`AES-256`	4	14	25.38	30.40
`Rijndael-256-256`	1	14	74.81	76.25
`Rijndael-256-256`	2	14	81.50	86.17

respectively, on the Apple Silicon M3 machine.

With the tools described in [Lem21, Lem23] we obtain that, on the Apple Silicon M3 machine, the 256-bit ciphers run at a sustained rate of approximately 2.8 CPU instructions per cycle, and the 512-bit ciphers at a rate of approximately 4.5 CPU instructions per cycle. This suggests that processing two 256-bit blocks in parallel with the same key would obtain a similar throughput than the 512-bit cipher for the same number of rounds.

5.1.2 Discussion of the Results

The first observation is that `Vistrutah`-256 is the best performing 256-bit wide cipher designed around the `AES` instructions in software implementations, as can be seen in Table 4. In fact, 12-round and 14-round `Vistrutah`-256 are even competitive with `Rijndael`-256-256 *with precomputed key schedules*. The latency of `Vistrutah`-512 is roughly twice that of `Vistrutah`-256 for the same number of rounds, as expected, but on microarchitectures with even more parallelism this gap can be reduced. Thus, it is competitive with an implementation of `Rijndael`-256-256 processing two blocks simultaneously. This means that the throughput of `Vistrutah`-512-256 would be the same as `Vistrutah`-256, while offering higher security margins and improved diffusion. An 18-round `Vistrutah`-512 would be slower, but it offers a security level of 512 bits.

`Vistrutah`-256 is significantly faster than `AES`-256 with inline key schedule up to 16 rounds. This is a distinction `Pholkos` enjoys only up to 12 rounds. `Rijndael`-256-256 with inline key schedule performs poorly. This behavior can be explained by how modern CPU microarchitectures handle the stack. Besides the fact that stack accesses have very good locality (hence the stack is always in the Level 1 cache), some microarchitectures have a very small "Level 0" cache dedicated to the stack, with speculation and renaming support. For small leaf functions, the stack data may never be written to memory, improving performance. The performance degradation can easily be explained if the size of the key expansion for `Rijndael`-256-256 (i.e., 30 128-bit values) exceeds the size of this cache.

Regarding `Ghidle`, in agreement with the claims made by its designers, we find that `Ghidle`-512 is faster than `Pholkos`-512, but `Ghidle`-256 is slower than `Pholkos`-256.

Finally, we compared Vistrutah to Simpira v2 [GM16] using their recommended round numbers. Our implementation is derived from the Simpira v2 source code and inherits its fully unrolled structure, which again does not work to our advantage. The Feistel function of Simpira was optimized specifically for the Intel architecture. This made sense at the time, given Intel's apparently unassailable market dominance. As a result, on NEON architectures, the Feistel function requires an additional explicit XOR, and setting the round constants is less efficient than on AVX. On NEON, Vistrutah-256 and Simpira-2 show comparable performance, with 10 rounds of Vistrutah-256 consistently outperforming Simpira-2. On AVX, Simpira-2 is faster. However, Simpira uses a Luby-Rackoff construction with 128-bit branches and a 128-bit key, limiting its security to 128 bits. This makes direct comparison difficult with Vistrutah-256, which has a longer key and potentially better resistance against Q1 quantum adversaries. The comparison between Simpira-4 and Vistrutah-512 yields similar results. However, recall that Vistrutah-512 offers security levels up to 512 bits.

5.2 A word about Hardware

Vistrutah was designed for software performance on platforms with AES instructions. However, it would have limited usefulness if its hardware performance were a regression compared to Rijndael-256-256. For this reason, we analyzed its hardware performance. For simplicity, we compare side-channel unprotected implementations with fully parallelized S-Box and MixColumns layers. We assume combined S-Boxes supporting both encryption and decryption, and circuits built from 1-input and 2-input gates only. Because of the use of the same building blocks, we expect that any additional optimisation would result in similar gains for all algorithms.

The two high-level components of an AES-derived cipher are the round function and key schedule, and each consists of a layer of S-Boxes combined with either a MixColumns layer or a sequence of XORs. We note the following:

- Several implementation options are available for the S-Box with an area of about 300 GE and a depth of 10.5 to 16 XORs (21 to 32 GE), such as [Can05, BP12, ME19, AR21]. Faster circuits exist (based on Look-Up Tables [GC09], Binary Decision Diagrams (BDD) [Jr.78], Twisted BDDs [MS02], or Positive Polarity Reed-Muller expansions [Sas93], cf. [CHCL25] for a detailed discussion of these approaches), but they are also much larger, restricting their usefulness.

- On one column of the AES state, the MixColumns operation is defined as

$$\begin{bmatrix} r'_0 \\ r'_1 \\ r'_2 \\ r'_3 \end{bmatrix} = \begin{bmatrix} x & x+1 & 1 & 1 \\ 1 & x & x+1 & 1 \\ 1 & 1 & x & x+1 \\ x+1 & 1 & 1 & x \end{bmatrix} \cdot \begin{bmatrix} r_0 \\ r_1 \\ r_2 \\ r_3 \end{bmatrix} .$$

  The multiplication by x is understood in the field $\mathrm{GF}(2^8)$, defined by the polynomial $x^8+x^4+x^3+x+1$, and is implemented with three 2-input XOR gates. MixColumns can thus be realized as 108 XORs as follows: $t_0 = r_0 + r_1$, $t_1 = r_1 + r_2$, $t_2 = r_2 + r_3$, $t_3 = r_3 + r_0$, followed by $r'_0 = 2 \cdot t_0 + t_2 + r_1$, $r'_1 = 2 \cdot t_1 + t_3 + r_2$, $r'_2 = 2 \cdot t_2 + t_0 + r_3$, $r'_3 = 2 \cdot t_3 + t_1 + r_0$, resulting in a circuit with a depth of 3 XOR gates. The area can be reduced, the best result to date being 92 XORs [Max19], but these methods all have a longer critical path of 6 XORs or higher. Since the area savings are minor, we use the more straightforward approach with the lower latency.

- If we approximate an S-Box, resp., the per-cell cost of MixColumns, with an area of 300 GE, resp. 50 GE and a delay of 25, resp., 6 GE, we have that a MixColumns layer has an area of one sixth of an S-Box layer, and one quarter of the latency.

- One round of the AES key schedule function consists of a layer of 4 S-Boxes, followed by a sequence of four XORs of 32-bit words, and a cell shuffle. Note that the data obfuscation path and the round key schedule run in parallel. Even though the latency of the linear part of the key schedule is 4 XORs, therefore higher than the 3 XORs of the MixColumns layer, we observe that the data obfuscation path contains also a round key addition. Hence, the critical path of the key schedule is not longer than the data obfuscation's.

- One round of the Rijndael-256-256 key schedule function consists of a two layers of 4 S-Boxes, each followed by 16 XORs with a latency of 8 GE, and a cell shuffle.

In Table 6, we compare the area and delay of ciphers considered in this paper using our approximations of S-Box and MixColumns layer costs. If larger and faster S-Boxes were used instead (up to twice as fast, but with areas 2-6 times larger), the relative area and delay ratios would change only minimally. Our analysis includes key expansion circuits. We note that a single key expansion circuit could serve multiple parallel data encryption paths using the same key. This would reduce the average area gap between Rijndael-256-256 and other evaluated ciphers, though only in specialized use cases.

The main observation is that Vistrutah's advantage in latency is reduced in HW implementations, since the more expensive key schedules can be performed in parallel to the main encryption path, but this also translates in a lower area. So the advantage is a different type of tradeoff. Simpira's high delay in HW should not come as a surprise, since its critical path contains a higher number of dependent AES rounds — a striking contrast to its speed in SW because its Feistel round function only contained AES functions and no special mixing layers, nor a key schedule.

In light of these results, one may ask why to search for a permutation that is efficient in SW, when, if the cipher is deployed (or even standardized), new hardware and an ISAs extension to support it would become available in a short time anyway. The answer is twofold. First, we must support existing platforms. Second, it is plausible that we may have ended up comparing a variant of Vistrutah with a fast state shuffle to a version with a mixing layer which is slower in software and gives rise to similar cell counts in fewer rounds — and that therefore would perform better in hardware. However, looking

Table 6: Approximate area and delay of the ciphers considered in this paper in HW. For a fair comparison, we removed the tweak from Pholkos and ASURA.

Cipher	Rounds	Block Size	Area (S-Boxes)	Delay (GE)	Relative Area	Relative Delay	A×D/s Factor
AES-128	10	128	237.7	326	1	1	1
AES-256	14	128	339.4	458	1.4	1.4	1.96
Rijndael-256-256	14	256	678.8	458	2.81	1.4	1.96
Pholkos-256	16	256	621.0	524	2.57	1.61	2.07
Pholkos-512	20	512	1554.3	656	6.42	1.4	2.25
Ghidle-256	21	256	754.6	635	3.12	1.95	3.04
Ghidle-256	24	256	862.2	726	3.56	2.14	3.81
Ghidle-512	24	512	1745.1	726	7.21	2.14	3.86
ASURA	12	256	464.9	392	1.92	1.2	1.16
Simpira-2	15×2	256	1132.8	1890	4.68	5.8	13.58
Simpira-4	15×2	256	2265.6	1890	9.37	5.8	13.58
Vistrutah-256 Full	14	256	542.9	458	2.24	1.4	1.56
Vistrutah-256 Short	10	256	386.8	326	1.59	1	0.79
Vistrutah-512 Full	18	512	1398.2	590	5.78	1.8	2.60
Vistrutah-512 Short	12	512	929.7	392	3.84	1.2	1.15

Table 7: Summary of the cryptanalysis of round-reduced Vistrutah. r = number of rounds, CP/ACC = chosen plaintexts/adaptively chosen ciphertexts. * = attack starts from Round 2.

			Data		Memory	
Attack type	r	Time	CP	ACC	(states)	Reference
Key-recovery attacks on Vistrutah-256						
Integral	7	$2^{135.2}$	$2^{130.6}$	–	$2^{46.6}$	Subsubsection E.1.1
Multiple-of-2	7	$2^{132.1}$	2^{128}	–	2^{129}	Subsubsection E.6.1
Impossible Diff.	7	2^{131}	2^{79}	–	2^{112}	Subsubsection E.2.1
Yoyo	8	$2^{202.4}$	$2^{65.9}$	$2^{97.9}$	$2^{98.9}$	Subsubsection E.5.1
Mixture	8	2^{193}	2^{160}	–	2^{163}	Subsubsection E.4.1
Integral	8	$2^{185.1}$	$2^{132.5}$	–	2^{128}	Subsubsection E.1.2
DS-MitM	8	$2^{170.6}$	2^{126}	–	2^{136}	Subsubsection E.7.1
Impossible Diff.	8	$2^{162.2}$	$2^{156.4}$	–	2^{144}	Subsubsection E.2.2
Boomerang	8*	$2^{130.1}$	$2^{95.5}$	$2^{97.6}$	$2^{97.5}$	Subsubsection E.3.1
Impossible Diff.	9	$2^{218.2}$	2^{210}	–	2^{176}	Subsubsection E.2.3
DS-MitM	10	$2^{255.0}$	2^{253}	–	$2^{252.5}$	Subsubsection E.7.3
Distinguishers for Vistrutah-512-256						
Multiple-of-8	10*	$2^{132.1}$	2^{128}	–	2^{129}	Subsubsection E.6.2
Mixture	10*	$2^{130.5}$	$2^{128.6}$	–	2^{130}	Subsubsection E.4.2
Key-recovery attacks on Vistrutah-512-256						
Impossible Diff.	10*	$2^{280.1}$	$2^{257.6}$	–	2^{257}	Subsubsection E.2.5
Boomerang	12*	$2^{222.1}$	$2^{218.4}$	2^{195}	$2^{183.5}$	Subsubsection E.3.3
Integral	12*	$2^{166.8}$	$2^{132.3}$	–	2^{128}	Subsubsection E.1.4
Distinguishers for Vistrutah-512						
Multiple-of-8	10*	$2^{132.1}$	2^{128}	–	2^{129}	Subsubsection E.6.2
Mixture	10*	$2^{130.5}$	$2^{128.6}$	–	2^{130}	Subsubsection E.4.2
Mixture	12*	$2^{368.2}$	2^{365}	–	2^{368}	Subsubsection E.4.3
Key-recovery attacks on Vistrutah-512						
DS-MitM	8	$2^{171.6}$	$2^{166.2}$	–	$2^{103.2}$	Subsubsection E.7.2
Impossible Diff.	10	$2^{299.0}$	$2^{259.8}$	–	2^{257}	Subsubsection E.2.4
Yoyo	10*	$2^{159.8}$	$2^{33.1}$	$2^{68.9}$	$2^{68.9}$	Subsubsection E.5.2
DS-MitM	12*	$2^{414.6}$	2^{413}	–	$2^{324.4}$	Subsubsection E.7.4
Integral	12*	$2^{252.9}$	2^{133}	–	2^{128}	Subsubsection E.1.3
Boomerang	12*	$2^{222.1}$	$2^{218.4}$	2^{195}	$2^{222.8}$	Subsubsection E.3.2
Impossible Diff.	14*	$2^{505.3}$	2^{424}	–	2^{439}	Subsubsection E.2.6

at Tables 2 and 3, this has only happened for $s = 4$ at the 307-bit security level (i.e., 256 bits plus 20 %), due to a careful choice of components. Therefore, we posit that this may be in general unlikely.

6 Summary of the Cryptanalysis of Vistrutah

We summarize the results of our cryptanalysis in Table 7. A high-level description of the attacks follows. Detailed technical contributions can be found in Appendix E.

6.1 Diffusion and Structure

Full diffusion is achieved in Vistrutah in just a few rounds. For Vistrutah-256, we have:

- Starting from a state after an even number of rounds, i.e., after a full step, a single bit will fully affect the state linearly after three rounds in encryption direction and nonlinearly after the subsequent `SubBytes` operation, and nonlinearly after three full rounds in decryption direction.
- Starting from a state after an odd number of rounds i.e., in the middle of a step, a single bit will fully affect the state after three full rounds in both directions.

For `Vistrutah`-512, we have:

- Starting from a state after an even number of rounds, i.e., after a number of full steps, a single bit will fully affect the state linearly after three rounds in encryption direction and nonlinearly after the subsequent `SubBytes` operation. In decryption direction, it will affect every bit of the state after three rounds and another inverse `MixColumns` operation linearly, and nonlinearly after four full rounds.
- Starting from a state after an odd number of rounds, i.e., after the first round in a step, a single bit will fully affect the state after four rounds in encryption direction linearly and nonlinearly after the subsequent `SubBytes` operation and after three inverse rounds in decryption nonlinearly.

From this, we can determine the maximum length of various distinguishers and key-recovery attack phases. It is important to note that key-recovery phases cannot always extend distinguishers to their maximum potential number of rounds, because most distinguishers require complex starting configurations beyond a simple single active byte.

The state transformation of `Haraka`-512 and `Pholkos`-512 structurally resembles the `AES`, but with 32-bit cells (the columns of the slices) instead of eight-bit cells. Consequently, when disregarding key schedule and specifics of the S-box, many attack techniques that work against the `AES` can be upscaled to `Haraka`-512 and `Pholkos`-512, by mapping bytes to columns, columns to state slices, with the attack complexities increasing proportionally with the larger state size. In particular, this applies to boomerang and mixture distinguishers.

6.2 Isomorphism between the `AES` and `AESQ`-like Transforms

Daemen et al. [DLP+09] defined four-round `AES`, the concatenation of two Super-boxes (where a Super-box is comprised of $\mathsf{SR} \circ \mathsf{MC} \circ \mathsf{SR}$) with a linear layer in between them, as a Mega-box. In their proposal of PAEQ and its underlying `AES`-based permutation `AESQ` in [BK14], Biryukov and Khovratovich identified an isomorphism between its structure and that of the `AES`: we can define a bijection $\varphi : \mathbb{Z}_{2^{16}} \to \mathbb{Z}_{2^{16}}$ that maps the byte-activity pattern of `AES` states to the diagonal-activity pattern of `AESQ`-like designs and activity transitions are preserved (with decreased probabilities for the larger-state transforms). An `AES` round, the mixing layer, and another `AES` round correspond to the application of $4 \cdot 4$ Super-boxes. Similarly, four-round `AESQ` can be represented as four parallel Mega-boxes.

`Vistrutah`-512 preserves the isomorphism of input diagonals to columns to inverse diagonals through a Super-box for the `AES` to the set of four input diagonals to a step, to columns that are mapped to rows of the same slice through mixing layer to the step outputs mapped by another mixing layer to the i-th row in each state slice. As long as the mixing layer is a permutation $\pi : \mathbb{Z}_{64} \to \mathbb{Z}_{64}$ such that it maps (i) the columns of each input slice to pairwise distinct output slices, and (ii) each input column's bytes to pairwise distinct different input diagonal in its output state slice, the isomorphism holds. There exist $(4!)^4$ permutations defining what input columns are mapped to what output slices and $(4!)^5$ permutations of what input bytes are mapped to what output positions in an output state slice. Over all four output slices, this produces a set of $(4!)^4 \cdot (4!)^{5 \cdot 4} = (4!)^{24} \approx 2^{110.04}$ isomorphism-preserving byte permutations, which includes the mixing-layer permutations of `AESQ`, `Haraka`, `Pholkos`, and `Vistrutah`-512.

Consequently, most structural distinguishing attacks on the AES-128 can be upscaled to Vistrutah-512, starting from the beginning of an even-indexed round (i.e., the middle of a step). Under suitable assumptions, many existing key-recovery attacks can be upscaled accordingly. A few limitations exist, for example, attacks that involve state or key guessing, such as Demirci-Selçuk MitM attacks, can be less effective as the adversary has to guess more variables since the isomorphism maps S-boxes in the AES to keyed Super-boxes in Vistrutah. We tested the most relevant attack types and obtained the results in Table 7. In the following, we summarize our findings on the most relevant types of attacks.

6.3 Differential and Linear Cryptanalysis

While we have assessed the resistance of Vistrutah against differential and linear cryptanalysis in Subsection 4.2, we still have to study the most relevant forms of impossible, boomerang, mixture, and yoyo differential, as well as MitM and slide attacks.

6.4 Integral Cryptanalysis

On Vistrutah-256, one can find integral trails [DKR97] on up to six rounds without and on seven rounds with the full codebook. We found that extending a distinguisher on $2+3$ rounds can be used effectively in an attack that appends three key-recovery rounds. This yields an eight-round attack with 2^{128} *Chosen Ciphertext*s (CCs). Using the *Fast-Fourier Transform* (FFT) approach [TA14], the partial-sum approach [FKL+00], or combinations of both [DGK+24] allows the attack complexities to remain below the time and data limits of 2^{256} *Encryption Equivalent*s (EEs) and queries.

On Vistrutah-512, both locations of a single-byte difference — after an even or after an odd number of rounds — allow for distinguishers on up to seven rounds without the full codebook, or on up to eight rounds with it. Using a diagonal difference allows for an eight-round integral distinguisher from four input diagonals, reflecting the distinguisher on four-round AES. From the isomorphism, the existing results on the AES [SLG+16, YTQ23] and the known links between impossible-differential, integral, and zero-correlation distinguishers [SLR+15], it seems that five-step distinguishers can be ruled out.

Integral attacks are relevant because the six-round attacks [TA14, DGK+24] that guess through the final two rounds possess the lowest complexity of all attacks on the AES. We can mount a straightforward adaptation to append a key-recovery procedure on four rounds to the end of the distinguisher. This results in a 12-round attack with 2^{133} *Chosen Plaintext*s (CPs) using the FFT approach. While we can envision that attacks may improve on our preliminary cryptanalysis by exploiting, e.g., details in the key-recovery procedure, we expect integral attacks to be limited to 10 rounds for the 256-bit version. For Vistrutah-512, there may be an attack on seven steps but with a complexity close to the full codebook, similarly as for the AES.

6.5 Impossible-differential Cryptanalysis

Impossible differentials [BBD+98, Knu98] allow to filter keys that would produce differential trails that are impossible to occur through a subcipher. We found impossible-differential distinguishers on $3+3$ rounds of Vistrutah-256 and $4+4$ rounds of Vistrutah-512. For Vistrutah-256, the mixing layer between steps is stronger than the column-wise blend operations in Haraka and Pholkos. The impossible-differential trail on three steps from Pholkos-256 [BLLS22] cannot be used as-is for Vistrutah-256, as the trail could have any arbitrary difference in one of the two slices during the first and third steps. However, a fully active difference in one slice at the beginning of the third step would be mapped to two active rows in all slices of the state by the inverse mixing layer. After the inverse

`MixColumns` operation, this could produce a fully active state, and an impossible transition in the middle cannot be guaranteed, at least not without considering the S-box details.

While $3 + 3$-round distinguishers can still be found, appending two rounds for key recovery proved significantly easier when only a single byte was active after five rounds of the distinguisher. We identified attacks on seven, eight, and nine rounds with increasing time complexities of 2^{131}, 2^{162}, and 2^{218} encryptions. The corresponding data complexities are 2^{79}, 2^{156}, and 2^{210} CPs. We cannot rule out that highly optimized (and sophisticated) attacks could reach 10 rounds of `Vistrutah`-256 but would expect such attacks to demand close to the full codebook.

The impossible-differential attack on seven-round `AES`-128 by Mala et al. [MDRM10] is known to be optimal for key recovery and has served as the basis for further cryptanalytic improvements [BNS14, BLNS18, LP21, LP25]. We adapted this attack to seven-step `Vistrutah`-512, resulting in enormous complexities of about 2^{500} EEs and *Memory Accesss* (MAs). The data complexity scales to approximately four times the exponent of the attack on the `AES`, reaching 2^{424} CPs. We also show a reduced attack on ten rounds with 2^{283} time complexity, which is close to the security goal of $n/2$ bit.

6.6 Boomerang Cryptanalysis

The fast diffusion in `AES`-like ciphers typically prevents long high-probability differential characteristics. For such ciphers, boomerang attacks [Wag99] can be particularly effective since they can combine multiple short differentials, such as in the breaks of full-round `AES`-192 and `AES`-256 in the RK setting [BK09, BKN09].

The two best attacks of this type on round-reduced `AES` in the single-key setting stem from recent works by Bariant et al. in [BL23] and [BDK+24] on six rounds. Among other forms of conditional differentials, mixture, boomeyong [RSP21], and boomerang-chain distinguishers [YTXQ24] all cover up to six rounds of `AES`-128 but have higher complexities than the six-round boomerang key-recovery attack from [BDK+24]. We adapted that attack to `Vistrutah`-512. It is a shifted-retracing boomerang approach [DKRS20, DKRS24], which waits for ciphertext pairs in low-probability subspaces and generates new pairs from them by XORing output differences to parts of their states that are equal for both ciphertexts. This generates a second candidate ciphertext pair that shares the same active-byte values in both texts during the lower decryption trail. Thus, if one pair follows the bottom differential trail, so does the other pair in the boomerang quartet. Moreover, filtering for the low-probability event of ciphertexts lying in a certain subspace increases the signal-to-noise ratio, and reduces the overall complexity by reducing the number of re-decryptions and *Adaptively Chosen Ciphertexts* (ACC) pairs.

Our 12-round attack achieves a time complexity of 2^{222} EEs, using 2^{218} CPs and 2^{195} ACCs. This represents the lowest complexity we found among all techniques on 12 rounds we could find. We explored extensions of the attack by one key-recovery round before or afterward. Prepending a round would require guessing three consecutive rounds of key material for proper filtering. This would need additional inactive-byte conditions on every decrypted pair to prevent guessing the full key. However, this makes it difficult or impossible to use multiple (mixture) ciphertext pairs whose inactive byte patterns follow from the first pair. While appending a round is possible by guessing only a subset of anti-diagonals in the final round, the resulting complexities exceeded that of an exhaustive key search in our studies. Careful improvements may reduce the time complexity or enable adding one or two rounds for key recovery, albeit at very high costs. However, due to the strong structural relation to `AES`, any substantially improved distinguisher would likely transfer to or from an improved attack on `AES` as well.

For `Vistrutah`-256, the different structure with only two state slices implies a lower probability of the truncated trails relative to the state size for the goal of having inactive slices in the trails, resulting in fewer steps that we can cover compared to the boomerang

on Vistrutah-512. We found an eight-round attack with time and data complexities slightly above 2^{130} EEs and below 2^{98} CPs/ACCs, respectively, which are the lowest among all eight-round attacks we could find. Two directions for improvement exist: adding one further intermediate step to the truncated trail or prepending a round for key recovery. In our studies, both attempts approached the complexity of 2^{256} encryptions. Therefore, we provide the eight-round attack in the appendix as a more descriptive result. While we can envision that careful attacks may cover up to 10 or even 11 rounds, these would likely have complexities close to or beyond the full codebook, and further extensions would require better techniques that will likely also be applicable in some form to AES.

6.7 Mixture-differential Cryptanalysis

Similar to boomerangs, mixture attacks [Gra18] are a variant of conditional-differential cryptanalysis. The adversary derives related pairs from an initial pair by exchanging (i.e., swapping) some active bytes. Given an initial pair whose ciphertext difference lies in a certain subspace, the goal is to exploit that the conditional probability for derived pairs to also lie in that (or a related) subspace is higher than for a random permutation. Grassi developed and formalized the concept for round-reduced AES as it was the structural property underneath the multiple-of-eight property. Ronjøm et al. [RBH17] formalized exchange functions; [BR19] generalized the concept to the exchange to further structures and proposed a longer distinguisher on up to six rounds of AES.

We can adapt the six-round exchange (or mixture) distinguisher by Bardeh and Rønjom [RBH17] to a distinguisher on 12 rounds of Vistrutah-512. As for our boomerang attack on 12 rounds, the procedure for Vistrutah-512 differs slightly in the first and last rounds, and the complexity of the resulting attack increases as expected to about a fourth power of that of the distinguisher on six-round AES, with approximately 2^{365} CPs and 2^{368} EEs. We could also mount a 10-round attack with lower complexities of ca. 2^{258} encryptions and 2^{255} CPs.

Mixture attacks require at least two independent active disjoint subcomponents of the plaintext that can be swapped to create mixed plaintext pairs. For that purpose, we can use the two slices of Vistrutah-256 as disjoint parts, with a truncated-differential trail that keeps the parts separate for up to three steps with probability below 2^{-256}. When reduced to seven rounds, the probability of having only eight active bytes in ciphertext differences for two related pairs drops below 2^{-320}. This compares favorably to the approximately 2^{-384} probability for a random permutation. We can append a round for key recovery and obtain an eight-round attack with 2^{160} CPs and 2^{194} MAs. We tried to extend this approach. While adding another step to the mixture trail is possible, its probability would fall below that for a random permutation. Future optimizations might enable attacks on up to 13 rounds for Vistrutah-512 and 11 rounds for Vistrutah-256, but their complexity would likely approach that of brute force.

6.8 Yoyo Cryptanalysis

The yoyo game in cryptanalysis is a close relative to boomerangs (and therefore mixtures). It was originally introduced by Biham et al. [BBD+98] as an adaptive re-decryption and re-encryption game. Ronjøm et al. [RBH17] transferred the setting to the AES and obtained the lowest-complexity adaptive distinguishers on four and five rounds; the complexity of their six-round distinguisher was later re-evaluated in [MRSA23] and found to slightly exceed that of an exhaustive search.

We transform their four-round distinguisher to the variants of Vistrutah, and find an adaption to eight-round Vistrutah-256, however, with slightly higher time complexity than the best mixture attack on eight rounds. For Vistrutah-512, we can transform a variant of the previous distinguisher on five-round AES into an attack with 2^{160} encryptions

and less than 2^{70} ACCs. Moreover, we also expect the 10-round boomeyong attack from [RSP21] to be directly transferable to 10-round `Vistrutah`-512. If the complexity of the six-round yoyo on `AES` can be lowered in the future, it is potentially also applicable to 12 rounds of `Vistrutah`-512. Stronger attacks than that are likely to require also technically improved attacks on the `AES`.

6.9 Multiple-of-n Cryptanalysis

Before their work on mixtures, Grassi et al. [GRR17] analysed the multiple-of-n property for five-round `AES`. Given a structure of plaintexts iterating over all possible values in one diagonal and leaving all other plaintext bytes constant, the number of ciphertext pairs in a subspace with $k \geq 1$ inactive inverse diagonals after almost five rounds (without the final `MixColumns` operation) is always a multiple of eight. Each such ciphertext pair originates from a structure with a single active column after one round, and there exist seven further pairs with those active bytes mixed that all share the same difference after three rounds. Since any pair with the i-th inverse diagonal being inactive after almost five rounds must stem from a pair with the i-th diagonal being inactive after three rounds, this guarantees that all eight pairs in such a structure have the same fully inactive inverse diagonals after almost five rounds.

We can apply the multiple-of-8 distinguisher directly to `Vistrutah`-512. This gives us a five-step distinguisher on Rounds 2–11, starting from a structure with the same i-th diagonal active in all state slices. The attack requires 2^{128} CPs and a small multiple of the complexity for encrypting them. For `Vistrutah`-256, we found a multiple-of-2 distinguisher on seven rounds that exploits the two parallel slices with similar time and data complexities as for `Vistrutah`-512.

6.10 Demirci-Selçuk Meet-in-the-middle Cryptanalysis

Demirci and Selçuk *meet-in-the-middle* key-recovery attacks (DS-MitM attacks, hereafter) [DS08] build upon an observation by Gilbert and Minier [GM00]: for a set of 256 related texts differing in a single byte (a δ-set), the number of possible value sets they can take through a few rounds depends only on the bytes that influence it on its path, e.g. at most 25 bytes from one byte to one byte through four rounds. This leaves at most $2^{25 \cdot 8}$ value sets that can be stored in a precomputed table instead of the $2^{256 \cdot 8}$ options theoretically achievable by a random permutation. Dunkelman et al. [DKS15a] further reduced the overall complexity by means of *differential enumeration*. The adversary first waits for a pair with a low-weight output difference, which implies that fewer variables have to be considered for creating the table of all pairs and a set of texts related to the pair (this is usually a δ-set, taking on all values in an active input byte and constant in all others). Thereupon, the adversary first guesses the keys for all candidate pairs and looks up if the resulting states are stored in the offline table. Derbez et al. [DFJ13, DF13] observed that the number of variables was lower than Dunkelman et al. estimated and exhausted the best attack configurations for all versions of the `AES`. As of today, Derbez et al.'s attack still possesses the lowest time complexity of attacks on `AES`-128 in the single secret-key model. Subsequent works improved the results on other variants and ciphers. Thus, DS-MitM attacks must be considered for any `AES`-like cipher. Sun [Sun21] noted that the length of an offline differential distinguisher for such attacks is upper bounded (for key lengths up to the block size) by twice the number of rounds needed for full diffusion. Thus, distinguishers can cover at most six and eight rounds for `Vistrutah`-256 and -512, respectively.

An important question is whether the best attack on `AES`-128 on seven rounds [DFJ13] could be transferred to `Vistrutah`-512. The key addition between the two rounds in each step is a crucial difference between the `AES` and `Vistrutah`-512, as DS-MitM attacks have to guess the internal state(s) in their offline phase. Since an S-box in the `AES` corresponds

to a keyed Super-box in Vistrutah-512, this implies the need to guess the full state in the middle and therefore the full key in the offline phase. The key-schedule equations could be used to reduce the amount of offline guessing to that of exhaustive search. Our analysis found $56 + 64$ guessed byte variables and 56 key-schedule constraints when fixing input and output differences of the distinguisher. While no direct attack was found, optimizations might reduce complexity below full codebook size by storing partial trails. Instead, we studied an attack that could use an offline distinguisher covering $3 + 3$ rounds from a single to a single active byte that required the guessing of significantly fewer bytes. That configuration allowed for extensions by three further rounds in both directions to a 12-round attack with data complexity close to the full codebook. We also found an attack on eight rounds with only 2^{128} CPs and $2^{171.6}$ EEs.

For Vistrutah-256, we found an attack with an offline distinguisher covering $3 + 3$ rounds which could be extended by two rounds at both sides to a 10-round attack with 2^{255} encryptions and 2^{253} CPs. Here, the probability of the ciphertext difference could not be increased easily by truncation as for the 512-bit version as those would add to the number of active output bytes and would require more bytes to be guessed in the middle step of the offline distinguisher. Using a four-round distinguisher, we found an attack on 8 rounds with 2^{126} CPs and $2^{170.6}$ encryptions. We can envision that further optimizations may lead to extensions by one more round for both versions. However, one can easily see that such attacks will have enormous complexities.

6.11 Slide Attacks

Slide attacks were described first by Biryukov and Wagner [BW99, BW00]. Improved several times since, e.g., [BBDK18, DKS15b, DKLS20], they exploit the fact that round functions and the key schedule produce equal states after different rounds. The nowadays established use of "pseudorandom" nothing-up-my-sleeve round constants derived from the binary expansion of the fractional part of π should effectively prevent such attacks. Moreover, Vistrutah's design is not tweaked. Thus, adversaries cannot cancel the influence of the different constants as they could by choosing tweaks.

6.12 Quantum Security

Attacks where the adversary can perform superimposed queries on an encryption or decryption oracle are out of scope. We do however consider attacks where the adversary has access to a large-scale quantum computer to analyze the output of classical queries. Against Grover search [Gro97] Vistrutah-256 offers at least 128 bits of security against quantum attacks, and Vistrutah-512 offers at least 256 bits of security against quantum attacks. Most likely, the actual security levels are significantly higher, as for the AES.

7 Security in the Context of Practical Settings

We now consider the strength of the design in relevant practical settings, i.e., contexts that occur naturally in widespread modes of operations, and which restrict the types of attacks that can be mounted. This allows us to determine the number of rounds required by the short versions of the cipher to maintain the security level expected by the mode.

7.1 Settings

Standard primitive cryptanalysis typically considers *Chosen-plaintext Attack* (CPA), CCA, and their adaptive variants, which, for block ciphers, are reflected in the *Pseudo-random Permutation* (PRP) and SPRP notions. Given the versatility of block ciphers, one is

naturally interested in primitives to satisfy security under those or even stronger settings, such as security under (chosen) key relations. But while using primitives that are secure in strong models simplifies the standardization and enhances flexibility, it may introduce some impractical overhead that can become costly in large-scale applications that do not need security under such strong artificial settings but have demanding throughput requirements. Efficiency concerns have motivated research on instantiating modes and schemes with round-reduced ciphers with a natural focus on the AES.

Given the bounds on the almost-XOR-universality of four-round AES [KS07], early designs such as MT-MAC-AES and PC-MAC-AES [MT06] could employ it in a provably secure manner. On the other hand, the ALRED designs and the related MACs ALPHA-MAC, [DR05a], Pelican-MAC, its successor [DR05b] and refinements [DR10], used four-round AES without a reductionist proof but with heuristic arguments. Hoang et al. coined the term "prove-then-prune" for the strategy of first proving the security of a mode before reducing the number of rounds of its internal primitives heuristically and applied it in AEZ [HKR15]. Several constructions followed that approach, in particular with round-reduced versions of the AES, e.g., AES-PRF [MN17], CTET$^+$ [CEL$^+$21], ForkCiphers [Ava13, ALP$^+$19], EliMAC [DMN23], ButterKnife [ACL$^+$24], and recently ZIP-AES [FGL$^+$24].

All of those build upon the observation that modes often restrict inputs to the internal primitive and protect its outputs from the attacker's direct observation and exploitation. However, security proofs of modes usually do not incorporate those aspects but replace the primitive at an early stage by an ideal counterpart instead. Thereupon, they bound the distance between both settings usually by the maximal PRP advantage of an adversary that issues the same number of calls as in the mode to a PRP or SPRP oracle. For example, the revised analysis of CENC$[w]$ [Iwa06, IMV16] showed a CPA advantage bound of

$$\mathbf{Adv}^{\mathsf{CPA}}_{\mathsf{CENC}[w]_{E_K}}(q,\ell,t) \leq \mathbf{Adv}^{\mathsf{PRP}}_{E_K}(\ell q(w+1)/w,t) + \frac{\ell w q}{2^n} \quad , \tag{26}$$

where q, ℓ, and t denote the number of queries, the number of blocks per query, and the time of the involved adversaries, and w is a rate parameter of CENC. While such an argument is sound and fine with a fully secure block cipher E_K, its coarseness ignores inherent protective features of the mode. Thus, results can become weak or even void if the primitive is replaced by a round-reduced version. For example, the theoretical security guarantees for CENC instantiated with six-round AES from Equation 26 would become meaningless for plausible choices of $q\ell = O(2^{48})$ as the PRP security of the latter can be broken with those resources [DGK$^+$24]. However, in practice, we are not aware of any attack on six-round AES in the context of CENC; we are not even aware of any attacks that would violate the security claims of Pelican-MAC 2.0 which employs only four-round AES. At that point, previous works that reduced the primitive either had to fall back on heuristic arguments (e.g. in the cases of AEZ or AES-PRF) or declared their new constructions as novel primitives themselves (as for ForkAES, ZIP-AES, or ButterKnife) and conducted wide-ranging cryptanalysis dedicated to the specific setting.[2]

Both approaches have their merits, in particular, as the mentioned works have pioneered research on more efficient constructions. However, they also have considerable disadvantages. On the one hand, the heuristic approach leaves users either without the guarantees of security proofs when instantiating modes with reduced-round primitives, or with "too much crypto" [Aum19] when using the full-round primitives. On the other hand, defining a new primitive for every new construction with round-reduced primitive calls would require a cumbersome and repetitive analysis every time, prohibiting the reusability of the analysis. Our work does not aim at defining new primitives or directions in provable security, but at finding sound ways of using Vistrutah and round-reduced versions in

[2]We note that ELiMAC based its security, in a similar fashion as this work, on a derived restricted variant of PRP security; in that case on that of seven-round AES evaluated under a random secret key and the (encoded) inputs $\{1,\ldots,2^{32}-1\}$.

widespread and relevant modes while still allowing meaningful security guarantees and allowing somewhat reusable but still mode-aware analysis of the primitive.

For the purpose of this work, we propose fine-grained primitive notions that allow us to analyze the security of our proposal in certain counter-based modes without having to define each of those settings as a new primitive. In total, we consider five settings, where the first two represent the standard SPRP and PRP notions as needed for various applications, plus three mode-aware notions motivated from variants of the fast, parallelizable, inverse-free, and therefore widespread *Counter mode* (CTR) mode and its use in schemes such as *Synthetic Initialization Vector* (SIV) [RS06], HCTR2 [CHB21], or *XOR of Random Permutations* (XORP) [Iwa06, IMV16] in bbb-ddd-AES [DMMT24].

- SPRP security: Security against (potentially adaptive) chosen plain-and-ciphertext adversaries that can choose inputs arbitrarily and observe the outputs. This is analogous to standard primitive cryptanalysis in the single secret-key model.
- PRP security: Security against (potentially adaptive) chosen-plaintext adversaries that can choose inputs arbitrarily and observe the outputs.
- wiPRP security: Weak iterated input-transform PRP security from random but known initial plaintexts, which are transformed with an affine update function to a sequence of block-cipher inputs, and observable corresponding ciphertext outputs. This reflects e.g. the usage of the block cipher in the CTR mode in widespread constructions such as SIV where the pseudorandom *Initial Value* (IV) is produced from the message by the PRF.
- wimPRP security: Weak iterated input-transform IV-masked PRP security from random but unknown plaintexts that are transformed with an affine update function to a sequence of block-cipher inputs, and observable corresponding ciphertext outputs. This reflects e.g. the usage of the block cipher in the CTR mode in HCTR2.
- wisPRP security: Weak iterated input-transform output-sum-only PRP security from random but unknown plaintexts that are transformed with an affine update function to a sequence of block-cipher inputs, with the corresponding outputs masked by adding a result from a random value generated with a distinct input from the primitive to each output block. This reflects, e.g., the usage of the block cipher in one chunk of XORP in bbb-ddd-AES.

Here, the prefix *weak* indicates that the adversary cannot choose the inputs to the primitive, in the spirit of the established weak PRF notions [NR95, NR98]. We illustrate the three latter settings in Figure 11 and provide the algorithmic definitions for the notions wiPRP, wimPRP, and wisPRP in Algorithm 2. In each of them, the challenger provides an oracle to a distinguisher $\mathbf{A}$. During the initialization, $\mathbf{A}$ provides a bijective input transform $f : \mathcal{M} \to \mathcal{M}$ and the challenger samples a random key K and a random bit that indicates if it will present either the real or the ideal world to $\mathbf{A}$. Subsequently, $\mathbf{A}$ can ask up to q queries of a positive integer ℓ^i each that indicates the number of output blocks. $\mathbf{A}$ is returned a keystream $Y_1^i \| \ldots \| Y_{\ell^i}^i$, which stems either from the construction with E_K in the real world or from the construction with a random permutation π in the ideal world instead. The notions differ in the aspects if $\mathbf{A}$ sees the internal IV, a masked version, or not, and if the keystream blocks are observable in plain or not. For all settings, the security of E_K is defined as the maximal distinguishing advantage of any adversary $\mathbf{A}$ that asks q queries of maximal length ℓ to the oracle and runs in time t.

As a result of those efforts, one could now define a variant of CENC$[w]$,[3] let us call it CENC$'[w]$, which masks its nonce with a random independent secret L, and guarantees a

[3]Note that for CENC, which expects a nonce, one can easily define a non-weak notion isPRP that replaces the IV by an input from $\mathbf{A}$.

Algorithm 2 The security games for the wiPRP, wimPRP, and wisPRP notions on a block cipher $E : \mathcal{K} \times \mathcal{M} \to \mathcal{M}$ that a distinguisher $\mathbf{A}$ plays with a challenger. $\mathbf{A}$ provides an input transform $f \in \mathsf{Perm}(\mathcal{M})$, can call wiPRPquery with inputs $\ell^i \in \mathbb{N}$ for $i \in \{1, \ldots, q\}$ and wins if it correctly guesses b at the end; init and finalize are the same for all notions.

procedure init(f)
  $\pi \xleftarrow{\$} \mathsf{Perm}(\mathcal{M})$
  $K \xleftarrow{\$} \mathcal{K}$
  $L \xleftarrow{\$} \mathcal{M}$
  $b \xleftarrow{\$} \{0, 1\}$

function finalize(b')
  if $b = b'$ **then return true**
  return false

function wiPRPquery(ℓ)
  $IV \xleftarrow{\$} \mathcal{M}$
  $U_0 \leftarrow IV$
  for $j \leftarrow 1..\ell$ **do**
    $U_j \leftarrow f(U_{j-1})$
    if $b = 1$ **then** $Y_j \leftarrow E_K(U_j)$
    else $Y_j \leftarrow \pi(U_j)$
  $\mathbf{Y} \leftarrow Y_1 \| \ldots \| Y_\ell$
  return $(IV, \mathbf{Y})$

function wimPRPquery(ℓ)
  $IV \xleftarrow{\$} \mathcal{M}$
  $U_0 \leftarrow IV \oplus L$
  for $j \leftarrow 1..\ell$ **do**
    $U_j \leftarrow f(U_{j-1})$
    if $b = 1$ **then** $Y_j \leftarrow E_K(U_j)$
    else $Y_j \leftarrow \pi(U_j)$
  $\mathbf{Y} \leftarrow Y_1 \| \ldots \| Y_\ell$
  return $(IV, \mathbf{Y})$

function wisPRPquery(ℓ)
  $IV \xleftarrow{\$} \mathcal{M}$
  $U_0 \leftarrow IV \oplus L$
  if $b = 1$ **then** $Y_0 \leftarrow E_K(U_0)$
  else $Y_0 \leftarrow \pi(U_0)$
  for $j \leftarrow 1..\ell$ **do**
    $U_j \leftarrow f(U_{j-1})$
    if $b = 1$ **then** $Y_j \leftarrow E_K(U_j) \oplus Y_0$
    else $Y_j \leftarrow \pi(U_j) \oplus Y_0$
  $\mathbf{Y} \leftarrow Y_1 \| \ldots \| Y_\ell$
  return $(IV, \mathbf{Y})$

Notation: $X \xleftarrow{\$} \mathcal{X} = X$ is sampled uniformly at random from a non-empty set $\mathcal{X}$; $\perp$ = invalid; $\mathsf{Perm}(\mathcal{X})$ = set of all permutations over $\mathcal{X}$.

security bound of

$$\mathbf{Adv}^{\mathsf{CPA}}_{\mathsf{CENC}'[w]_{E_K}}(q, \ell, t) \leq \mathbf{Adv}^{\mathsf{wisPRP}}_{E_K}(\ell q(w+1)/w, t) + \frac{\ell w q}{2^n} \quad .$$

Then, cryptographers can analyze the security of the (potentially round-reduced) primitive E under the wisPRP setting, which may be negligible even under demanding data allowances such as $q\ell = O(2^{100})$ and plausible high-throughput-oriented choices of $w = 16$ and six-round AES. That is, if no attacks on the primitive are found under the wisPRP notion, this approach now yields a meaningful reductionist security result while allowing efficient instantiations for E. It remains a heuristic systematic analysis, but now the cryptanalyst can focus on the relevant parts of the mode for evaluating the security of the primitive.

In the following, we summarize our findings for all versions of Vistrutah in these settings. The numbers of rounds we find (or consider likely to be) attackable are also summarized in Table 8.

7.2 Security of Round-reduced Vistrutah in the SPRP Setting

In this setting, all attacks on Vistrutah-512 from Table 7 apply. First, we consider a security level of k bit. For Vistrutah-512, structural attacks that do not exploit the key schedule benefit from the existing analysis of AES-128 due to the isomorphism between the two constructions. Consequently, the best structural attacks against AES-128 largely transfer to Vistrutah-512, though with some modifications. For Vistrutah-256, we identified DS-MitM attacks effective against up to 10 rounds (potentially extendable to 11), along with impossible-differential, mixture, and integral distinguishers covering up to seven rounds. Our detailed preliminary analysis of this case is summarized in Section 6.

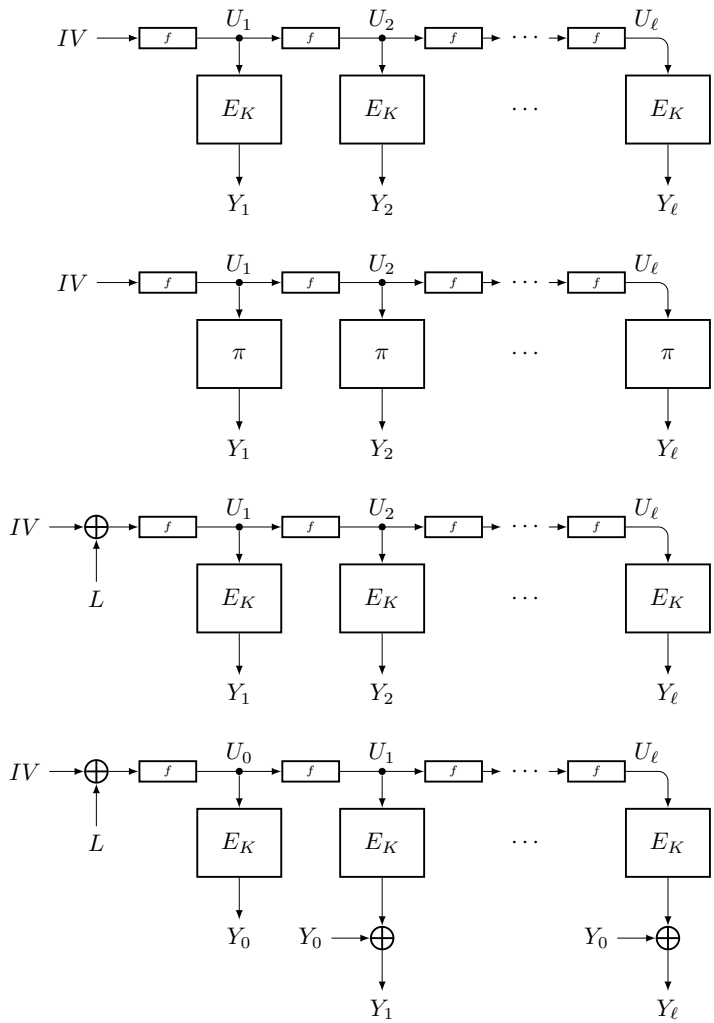


Figure 11: Computation flows in the mode-aware weak iterated input-transform PRP notions that we consider in this section. First figure from above: the real world in the weak iterated input-transform PRP (wiPRP) notion. Second figure: the ideal world in the wiPRP notion. Third figure: the real world in the IV-masked variant of the notion (wimPRP). Fourth figure: the real world in the output-sum-only variant of the notion (wisPRP).

Next, we consider an SPRP security security level of $n/2$-bit, i.e., as the limit for time, data, and memory complexity. For Vistrutah-512 and -512-256, the best key-recovery attacks we could identify in this setting are the integral and boomerang attacks which both still cover 12 rounds herein. In terms of distinguishers, the 12-round mixture distinguisher exceeds the complexities so that the remaining best distinguishers in this setting become the mixture and the multiple-of-8 distinguishers over 10 rounds.

Being a little generous for Vistrutah-256, the best key-recovery attack with about $2^{n/2}$ time, data, and memory we could identify is an eight-round boomerang attack with 2^{130} computations and 2^{95} CPs + 2^{97} ACCs; integral and impossible-differential distinguishers on six rounds also apply in this setting.

Table 8: Preliminary *results* and *estimations* on the number of rounds that attacks can cover for the individual variants of `Vistrutah` in different models used in practical modes of operation. Security in bits; •/– = feature is present/absent.

	Adversarial capabilities							#Rounds for security of					
			Inputs			Outputs		k bit			$n/2$ bit		
Setting	Decryption oracle	Adaptive	Choosable	Known	Known difference	Known	Known difference	512	512-256	256	512	512-256	256
SPRP security	•	•	•	•	•	•	•	14	12	10	12	12	8
PRP security	–	•	•	•	•	•	•	14	12	10	12	12	7
wiPRP security	–	–	–	•	•	•	•	12	11	10	11	11	7
wimPRP security	–	–	–	–	•	•	•	12	11	10	11	11	7
wisPRP security	–	–	–	–	•	–	•	12	10	7	10	10	7

7.3 Security of Round-reduced `Vistrutah` in the PRP Setting

For the `AES` as for the different variants, most attacks also apply in this, the common PRP, setting. The exceptions are boomerangs and yoyo attacks, which explicitly require adaptively chosen ciphertexts. The best key-recovery attack on `Vistrutah`-512 for k-bit security remains the impossible-differential attack on 14 rounds, and for $n/2$-bit security, this role is taken by the integral attack on 12 rounds. For $n/2$-bit security as well as for `Vistrutah`-512-256, the 12-round integral attack becomes the best attack and the 10-round mixture and multiple-of-8 become the best distinguishers.

For `Vistrutah`-256, the best attacks remain the 10-round DS-MitM and the nine-round impossible-differential attacks; only for $n/2$-bit security, the boomerang on eight rounds can no longer be applied.

7.4 Security of Round-reduced `Vistrutah` in the wiPRP Setting

Moving from chosen-plaintext to known-plaintext settings, the security of a primitive becomes more challenging to analyze, as direct control over plaintext inputs is lost. We examine attacks on primitives used in modes like CTR that utilize simple affine update functions. In this scenario: (i) The attacker knows the initial value but cannot choose it: it appears random from their perspective; (ii) This value may originate from encrypting a nonce; and (iii) The adversary can compute the input blocks after this initial value.

The plaintext-only restriction makes attacks requiring chosen ciphertexts (such as boomerang or yoyo attacks) difficult if not impossible to execute. Additionally, the requirement for random known plaintexts complicates obtaining input pairs with specific differences. Any attack relying on related inputs must leverage differences created by the update function, with effectiveness depending on what input relationships the function permits. If the update function allows sampling input sets of our integral distinguisher, as is the case e.g., if it based on a counter, integral attacks become the strongest attacks in this setting. For `Vistrutah`-512 and -512-256, this implies that attacks on 12 rounds may still be possible. However, the integral attack we show starts from values whose active diagonals are far apart; a simple counter would need $O(2^{416})$ blocks to sample the full set of the 12-round attack. However, it would require only 2^{128} iterations to sample the input set for the same attack with the first round removed and to conduct an 11-round attack. The 10-round multiple-of-8 and mixture distinguishers are thus the best ones we can use in this setting.

If the input update function makes sampling integral sets unobtainable in $O(2^{256})$

computations, we are left, e.g., with 10-round mixtures that start from random (but observable) differences in a 2^{384}-element space.

For Vistrutah-256, boomerang and yoyo attacks are also inapplicable in this setting. If the update function enables the sampling of input sets for an integral distinguisher, such a distinguisher could cover seven rounds. The nine-round impossible-differential attack remains possible but requires more data to find sufficient plaintext pairs in the restricted input space. Using the first 128-bit slice as the input-difference space with a simple counter, our nine-round impossible-differential attack still applies. Additionally, the DS-MitM attack on ten rounds could work since the counter must iterate through 2^{128} blocks per structure, approaching full codebook and maximal time complexity. For $n/2$-bit (128-bit) PRP security of Vistrutah-256, seven-round integral and impossible-differential attacks remain viable when using a basic counter update function.

7.5 Security of Round-reduced Vistrutah in the wimPRP Setting

This setting further weakens the adversary compared to the wiPRP setting. Again, the PRP condition renders all attacks that depend on chosen ciphertexts inapplicable, such as yoyo or boomerang attacks; the condition of random unknown initial values hides the plaintexts and their differences. Attacks based on related inputs have to exploit related differences from the update function, and their strength depends on what input relations the update function allows. Again, if the update function allows the adversary to sample input sets of an integral distinguisher, as is the case e.g., with an integer counter or a concatenation of a counter with a nonce or a random initial value, integral attacks remain applicable. For Vistrutah-512 and -512-256, this implies that attacks on 12 rounds could still be possible if they could be sampled in full as in our 12-round attack, or on 11 rounds if the update function is a naive counter. The best distinguisher we can find remains the 10-round multiple-of-8 one.

For Vistrutah-256, again, if the update function allows sampling input sets of an integral attack, such an attack can cover eight rounds. Moreover, the impossible-differential attack on nine rounds could still be applicable with higher data complexity as sufficiently many plaintext pairs in the input space could be deterministically and efficiently sampled by the update function, which a naive counter could guarantee, which could also allow the DS-MitM attack on ten rounds to still hold. For 128-bit PRP security, the integral and the impossible-differential attacks plus a little guessing could still apply to seven rounds.

For the case of related keys and inputs either fixed or related, see Subsection 7.7.

7.6 Security of Round-reduced Vistrutah in the wisPRP Setting

This setting weakens the adversary even further, as a random independent value is used for masking the outputs, as happens in modes such as XORP inside an accordion mode.

The encryption-only restriction renders all attacks that depend on chosen ciphertexts, such as yoyo or boomerang, hard or inapplicable. Plaintext values are unknown, and the only potentially known input relations are the differences between outputs from different iterations of the update function. The capabilities of the adversary thus depend on what the update function allows. On top of it, key guessing on both sides becomes difficult because the values are undisclosed. On the plaintext side, the values are unknown, so the key cannot be derived precisely without also guessing the state; on the output side, individual outputs from the primitive cannot be isolated and observed.

If the update function allows sampling input sets that enable mixture distinguishers or mixtures between different blocks of the same query (that are XORed with the same masking value), those become the best attacks covering up to 10 rounds for both Vistrutah-512 and Vistrutah-512-256. Obtaining all values of multiple-of-8 and integral distinguishers with the same masking values is usually impossible for plausible rates of XORP. However,

obtaining input differences to mount an eight-round impossible-differential distinguisher (scaled up from the four-round distinguisher on AES-128 [BK01]) appears also possible.

For Vistrutah-256, if the update function allows sampling sets of an integral or ID distinguisher, those could cover six or even seven rounds and appear as the best attacks we could find in this setting.

7.7 Key Agility

Vistrutah incurs no costs for re-keying, unlike AES, Rijndael-256-256, and Pholkos where re-keying is computationally expensive. This enables the design of modes of operation where each Vistrutah instance can use a different key.

In the ideal-cipher model, the mapping $K \mapsto E_K(X)$ for a fixed block X and a block cipher E is a PRF for close to 2^n queries, where n is the block size. However, this cannot be assumed for a concrete block cipher. Assuming that a version of Vistrutah enjoys RK security up to 2^j queries, we can assume in theory that the mapping $K \mapsto E_K(X)$ is a PRF up to 2^j queries, and if $j \geq n/2$, we obtain a BBB limit on the number of message blocks that can be encrypted with the same initial key and input X. This is conceptually similar to Counter-in-Tweak modes [PS16], provided that also X is independently sampled and the number of message block is also limited to be small with respect to 2^n. This said, relatively few efforts have been devoted to the security of such approaches, and any security assessment would require performing RK analogs of many attacks that so far have been studied only in the single-key model.

8 Conclusion

With Vistrutah we achieve something that until recently would have been considered an unlikely feat: designing a large block cipher with blocks of 256 and 512 bits that can offer superior security *and* comparable, if not better, performance than Rijndael-256-256. The cipher also possesses a more robust key schedule that prevents RK attacks, whereas AES-256 suffers from a significant security reduction under this model — for fairness, such attacks are completely unpractical anyway, but they prove that the key schedule of the AES is not as robust as the rest of the design. This is unsurprising since at the time of the development of the AES there was a much more limited understanding of key schedules.

14-rounds Vistrutah-256 and Vistrutah-512-256 offer 256 bits of security with reasonable margins. 18-rounds Vistrutah-512-512 offers 512 bits of security with reasonable margins. Our extensive cryptanalysis shows that these margins may be quite difficult to erode. Our contribution alone probably provides more, and more up-to-date cryptanalysis than what has been performed on Rijndael-256-256 so far. An isomorphism between Vistrutah-512 and the AES, allows to transform essentially the entire cryptanalysis of AES into results about Vistrutah-512.

Short versions of Vistrutah — namely, to 10 and 12 rounds, respectively, for the 256- and 512-bit variants — have been designed for use in the keystream generation part in HEH modes of operation like HCTR2, starting with an encrypted initialization vector.

Vistrutah adopts a novel concept of alternating a fixed round key with a variable one in the step function. This achieves two goals: (i) it reduces the number of inverse MixColumns operations that would be otherwise required; and (ii) it changes the relations needed for cancellations between round keys at each step. This seems to have been the most significant idea that contributed to Vistrutah's resistance against RK attacks.

Vistrutah is highly performing in software, performing almost always better than all other AES round function based alternatives, with few exceptions. Hardware implementations are also highly efficient, with execution times similar to Rijndael-256-256 but lower area, and thus lower power consumption.

We encourage cryptanalysis of ForkCipher schemes [Ava13, ALP+19] around Vistrutah and of ZIP-Vistrutah PRF analogues of ZIP-AES [FGL+24]. While we discussed how to use the short versions of Vistrutah in such contexts, an analysis of the resulting constructions is outside the scope of the present paper.

Acknowledgements

We are grateful to the anonymous reviewers and the shepherd for their comments and suggestions. Bishwajit Chakraborty was supported by the NRF-ANR project SELECT ("NRF2020-NRF-ANR072"). Eik List has been partially supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) project no. 509754807 and the Ministry of Education in Singapore under Grant RG102/24.

References

[AA20] Siavash Ahmadi and Mohammad Reza Aref. Generalized Meet in the Middle Cryptanalysis of Block Ciphers With an Automated Search Algorithm. *IEEE Access*, 8:2284–2301, 2020. doi:10.1109/ACCESS.2019.2962101.

[ABD+23] Roberto Avanzi, Subhadeep Banik, Orr Dunkelman, Maria Eichlseder, Shibam Ghosh, Marcel Nageler, and Francesco Regazzoni. The QARMAv2 Family of Tweakable Block Ciphers. *IACR Transactions on Symmetric Cryptology*, 3:25–73, 9 2023. doi:10.46586/tosc.v2023.i3.25-73.

[AC11] Martin R. Albrecht and Carlos Cid. Cold Boot Key Recovery by Solving Polynomial Systems with Noise. In Javier López and Gene Tsudik, editors, *Applied Cryptography and Network Security - 9th International Conference, ACNS 2011, Nerja, Spain, June 7-10, 2011. Proceedings*, volume 6715 of *Lecture Notes in Computer Science*, pages 57–72, 2011. doi:10.1007/978-3-642-21554-4_4.

[ACL+24] Elena Andreeva, Benoît Cogliati, Virginie Lallemand, Marine Minier, Antoon Purnal, and Arnab Roy. Masked Iterate-Fork-Iterate: A New Design Paradigm for Tweakable Expanding Pseudorandom Function. In Christina Pöpper and Lejla Batina, editors, *Applied Cryptography and Network Security - 22nd International Conference, ACNS 2024, Abu Dhabi, United Arab Emirates, March 5-8, 2024, Proceedings, Part II*, volume 14584 of *Lecture Notes in Computer Science*, pages 433–459. Springer, 2024. doi:10.1007/978-3-031-54773-7_17.

[AJN14] Jean-Philippe Aumasson, Philipp Jovanovic, and Samuel Neves. Analysis of NORX: Investigating Differential and Rotational Properties. In Diego F. Aranha and Alfred Menezes, editors, *Progress in Cryptology — LATINCRYPT 2014, Third International Conference on Cryptology and Information Security in Latin America, Florianópolis, Brazil, September 17-19, 2014, Revised Selected Papers*, volume 8895 of *Lecture Notes in Computer Science*, pages 306–324. Springer, 2014. doi:10.1007/978-3-319-16295-9_17.

[ALP+19] Elena Andreeva, Virginie Lallemand, Antoon Purnal, Reza Reyhanitabar, Arnab Roy, and Damian Vizár. Forkcipher: A New Primitive for Authenticated Encryption of Very Short Messages. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology — ASIACRYPT 2019. 25th International Conference on the Theory and Application of Cryptology and*

Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part II, volume 11922 of *Lecture Notes in Computer Science*, pages 153–182. Springer, 2019. doi:10.1007/978-3-030-34621-8_6.

[AR21] Doaa Ashmawy and Arash Reyhani-Masoleh. A Faster Hardware Implementation of the AES S-box. In *28th IEEE Symposium on Computer Arithmetic, ARITH 2021, Lyngby, Denmark, June 14-16, 2021*, pages 123–130. IEEE, 2021. doi:10.1109/ARITH51176.2021.00034.

[Arm23a] Arm. Arm® Cortex-X4 Core Software Optimization Guide, Issue 3.0, May 2023.

[Arm23b] Arm. Arm® Cortex®-A720 Core Software Optimization Guide, Issue 7.0, November 2023.

[Arm24] Arm. Arm Architecture Reference Manual for A-profile architecture, 2024. Available from: https://developer.arm.com/documentation/ddi0487/latest.

[Aum19] Jean-Philippe Aumasson. Too Much Crypto. *IACR Cryptology ePrint Archive*, 2019. Available from: https://eprint.iacr.org/2019/1492.

[Ava13] Roberto Avanzi. Method and apparatus to encrypt plaintext, US Patent US9294266B2, 2013.

[BA20] Elnaz Bagherzadeh and Zahra Ahmadian. MILP-based automatic differential search for LEA and HIGHT block ciphers. *IET Inf. Secur.*, 14(5):595–603, 2020. doi:10.1049/IET-IFS.2018.5539.

[BBD+98] Eli Biham, Alex Biryukov, Orr Dunkelman, Eran Richardson, and Adi Shamir. Initial Observations on Skipjack: Cryptanalysis of Skipjack-3XOR. In Stafford E. Tavares and Henk Meijer, editors, *Selected Areas in Cryptography '98, SAC'98, Kingston, Ontario, Canada, August 17-18, 1998, Proceedings*, volume 1556 of *Lecture Notes in Computer Science*, pages 362–376. Springer, 1998. doi:10.1007/3-540-48892-8_27.

[BBD+24] Arghya Bhattacharjee, Ritam Bhaumik, Nilanjan Datta, Avijit Dutta, and Sougata Mandal. BBB Secure Arbitrary Length Tweak TBC from n-bit Block Ciphers. *IACR Cryptology ePrint Archive*, 2024. Available from: https://eprint.iacr.org/2024/2049.

[BBDK18] Achiya Bar-On, Eli Biham, Orr Dunkelman, and Nathan Keller. Efficient Slide Attacks. *Journal of Cryptology*, 31(3):641–670, 2018. doi:10.1007/S00145-017-9266-8.

[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 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part III*, volume 12172 of *Lecture Notes in Computer Science*, pages 419–448. Springer, 2020. doi:10.1007/978-3-030-56877-1_15.

[BBS99a] Eli Biham, Alex Biryukov, and Adi Shamir. Cryptanalysis of Skipjack Reduced to 31 Rounds Using Impossible Differentials. In Jacques Stern, editor, *Advances in Cryptology — EUROCRYPT '99, International Conference on*

the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding, volume 1592 of *Lecture Notes in Computer Science*, pages 12–23. Springer, 1999. doi:10.1007/3-540-48910-X_2.

[BBS99b] Eli Biham, Alex Biryukov, and Adi Shamir. Miss in the Middle Attacks on IDEA and Khufu. In Lars R. Knudsen, editor, *Fast Software Encryption, 6th International Workshop, FSE '99, Rome, Italy, March 24-26, 1999, Proceedings*, volume 1636 of *Lecture Notes in Computer Science*, pages 124–138. Springer, 1999. doi:10.1007/3-540-48519-8_10.

[BCF+24] Ritam Bhaumik, André Chailloux, Paul Frixons, Bart Mennink, and María Naya-Plasencia. Block Cipher Doubling for a Post-Quantum World. *IACR Commun. Cryptol.*, 1(3):4, 2024. doi:10.62056/AV4FVUA5V.

[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. 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*, volume 7658 of *Lecture Notes in Computer Science*, pages 208–225. Springer, 2012. doi:10.1007/978-3-642-34961-4_14.

[BDF24] Christina Boura, Patrick Derbez, and Margot Funk. Alternative Key Schedules for the AES. In Christina Pöpper and Lejla Batina, editors, *Applied Cryptography and Network Security - 22nd International Conference, ACNS 2024, Abu Dhabi, United Arab Emirates, March 5-8, 2024, Proceedings, Part II*, volume 14584 of *Lecture Notes in Computer Science*, pages 485–506. Springer, 2024. doi:10.1007/978-3-031-54773-7_19.

[BDK+24] Augustin Bariant, Orr Dunkelman, Nathan Keller, Gaëtan Leurent, and Victor Mollimard. Improved Boomerang Attacks on 6-Round AES. *IACR Cryptology ePrint Archive*, page 977, 2024. Available from: https://eprint.iacr.org/2024/977.

[BJK+16] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY Family of Block Ciphers and Its Low-Latency Variant MANTIS. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology — CRYPTO 2016, 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*, volume 9815 of *LNCS*, pages 123–153. Springer, 2016. doi:10.1007/978-3-662-53008-5_5.

[BK01] Eli Biham and Nathan Keller. Cryptanalysis of Reduced Variants of Rijndael, 2001. Available from: http://csrc.nist.gov/archive/aes/round2/conf3/papers/35-ebiham.pdf.

[BK09] Alex Biryukov and Dmitry Khovratovich. Related-Key Cryptanalysis of the Full AES-192 and AES-256. In Mitsuru Matsui, editor, *Advances in Cryptology — ASIACRYPT 2009, 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings*, volume 5912 of *LNCS*, pages 1–18. Springer, 2009. doi:10.1007/978-3-642-10366-7_1.

[BK14] Alex Biryukov and Dmitry Khovratovich. PAEQ: Parallelizable Permutation-Based Authenticated Encryption. In Sherman S. M. Chow, Jan Camenisch, Lucas Chi Kwong Hui, and Siu-Ming Yiu, editors, *Information Security - 17th International Conference, ISC 2014, Hong Kong, China, October 12-14, 2014. Proceedings*, volume 8783 of *Lecture Notes in Computer Science*, pages 72–89. Springer, 2014. doi:10.1007/978-3-319-13257-0_5.

[BKN09] Alex Biryukov, Dmitry Khovratovich, and Ivica Nikolić. Distinguisher and Related-Key Attack on the Full AES-256. In Shai Halevi, editor, *Advances in Cryptology — CRYPTO 2009, 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings*, volume 5677 of *LNCS*, pages 231–249. Springer, 2009. doi:10.1007/978-3-642-03356-8_14.

[BKR11] Andrey Bogdanov, Dmitry Khovratovich, and Christian Rechberger. Biclique Cryptanalysis of the Full AES. In Dong Hoon Lurl and Xiaoyun Wang, editors, *Advances in Cryptology — ASIACRYPT 2011, 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings*, volume 7073 of *LNCS*, pages 344–371. Springer, 2011. doi:10.1007/978-3-642-25385-0_19.

[BL23] Augustin Bariant and Gaëtan Leurent. Truncated Boomerang Attacks and Application to AES-Based Ciphers. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023, 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part IV*, volume 14007 of *Lecture Notes in Computer Science*, pages 3–35. Springer, 2023. doi:10.1007/978-3-031-30634-1_1.

[BLLS22] Jannis Bossert, Eik List, Stefan Lucks, and Sebastian Schmitz. Pholkos - Efficient Large-State Tweakable Block Ciphers from the AES Round Function. In Steven D. Galbraith, editor, *Topics in Cryptology — CT-RSA 2022, Cryptographers' Track at the RSA Conference 2022, Virtual Event, March 1-2, 2022, Proceedings*, volume 13161 of *Lecture Notes in Computer Science*, pages 511–536. Springer, 2022. doi:10.1007/978-3-030-95312-6_21.

[BLNS18] Christina Boura, Virginie Lallemand, María Naya-Plasencia, and Valentin Suder. Making the Impossible Possible. *Journal of Cryptology*, 31(1):101–133, 2018. doi:10.1007/S00145-016-9251-7.

[BNS14] Christina Boura, María Naya-Plasencia, and Valentin Suder. Scrutinizing and Improving Impossible Differential Attacks: Applications to CLEFIA, Camellia, LBlock and Simon. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology — ASIACRYPT 2014, 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I*, volume 8873 of *Lecture Notes in Computer Science*, pages 179–199. Springer, 2014. doi:10.1007/978-3-662-45611-8_10.

[BP12] Joan Boyar and René Peralta. A Small Depth-16 Circuit for the AES S-Box. In Dimitris Gritzalis, Steven Furnell, and Marianthi Theoharidou, editors, *Information Security and Privacy Research - 27th IFIP TC 11 Information Security and Privacy Conference, SEC 2012, Heraklion, Crete, Greece, June 4-6, 2012. Proceedings*, volume 376 of *IFIP Advances in*

Information and Communication Technology, pages 287–298. Springer, 2012. doi:10.1007/978-3-642-30436-1_24.

[BR19] Navid Ghaedi Bardeh and Sondre Rønjom. The Exchange Attack: How to Distinguish Six Rounds of AES with $2^{88.2}$ Chosen Plaintexts. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology — ASIACRYPT 2019. 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 812, 2019, Proceedings, Part III* , volume 11923 of *Lecture Notes in Computer Science*, pages 347–370. Springer, 2019. doi:10.1007/978-3-030-34618-8_12.

[BS91] Eli Biham and Adi Shamir. Differential Cryptanalysis of DES-like Cryptosystems. *Journal of Cryptology*, 4(1):3–72, 1991. doi:10.1007/BF00630563.

[BW99] Alex Biryukov and David A. Wagner. Slide Attacks. In Lars R. Knudsen, editor, *Fast Software Encryption, 6th International Workshop, FSE '99, Rome, Italy, March 24-26, 1999, Proceedings*, volume 1636 of *Lecture Notes in Computer Science*, pages 245–259. Springer, 1999. doi:10.1007/3-540-48519-8_18.

[BW00] Alex Biryukov and David A. Wagner. Advanced Slide Attacks. In Bart Preneel, editor, *Advances in Cryptology — EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding*, volume 1807 of *Lecture Notes in Computer Science*, pages 589–606. Springer, 2000. doi:10.1007/3-540-45539-6_41.

[Can05] David Canright. A Very Compact S-Box for AES. In Josyula R. Rao and Berk Sunar, editors, *Cryptographic Hardware and Embedded Systems — CHES 2005, 7th International Workshop, Edinburgh, UK, August 29 — September 1, 2005, Proceedings*, volume 3659 of *LNCS*, pages 441–455. Springer, 2005. doi:10.1007/11545262_32.

[CB07] Nicolas T. Courtois and Gregory V. Bard. Algebraic Cryptanalysis of the Data Encryption Standard. In Steven D. Galbraith, editor, *Cryptography and Coding, 11th IMA International Conference, Cirencester, UK, December 18-20, 2007, Proceedings*, volume 4887 of *Lecture Notes in Computer Science*, pages 152–169. Springer, 2007. doi:10.1007/978-3-540-77272-9_10.

[CDL+20] Anne Canteaut, Sébastien Duval, Gaëtan Leurent, María Naya-Plasencia, Léo Perrin, Thomas Pornin, and André Schrottenloher. Saturnin: a suite of lightweight symmetric algorithms for post-quantum security. *IACR Transactions on Symmetric Cryptology*, 2020(S1):160–207, 2020. doi:10.13154/TOSC.V2020.IS1.160-207.

[CEL+21] Benoît Cogliati, Jordan Ethan, Virginie Lallemand, ByeongHak Lee, Jooyoung Lee, and Marine Minier. CTET+: A Beyond-Birthday-Bound Secure Tweakable Enciphering Scheme Using a Single Pseudorandom Permutation. *IACR Transactions on Symmetric Cryptology*, 2021(4):1–35, 2021. doi:10.46586/TOSC.V2021.I4.1-35.

[CGLS22] Debrup Chakraborty, Sebati Ghosh, Cuauhtemoc Mancillas López, and Palash Sarkar. FAST: Disk encryption and beyond. *Adv. Math. Commun.*, 16(1):185–230, 2022. doi:10.3934/AMC.2020108.

[CHB21] Paul Crowley, Nathan Huckleberry, and Eric Biggers. Length-preserving encryption with HCTR2. *IACR Cryptology ePrint Archive*, 2021. Available from: https://eprint.iacr.org/2021/1441.

[CHCL25] Phaedra S. Curlin, Jeff Heiges, Calvin Chan, and Tamara Silbergleit Lehman. A Survey of Hardware-Based AES SBoxes: Area, Performance, and Security. *ACM Comput. Surv.*, 57(9):229:1–229:37, 2025. doi:10.1145/3724114.

[CHNE24] Debasmita Chakraborty, Hosein Hadipour, Phuong Hoa Nguyen, and Maria Eichlseder. Finding Complete Impossible Differential Attacks on AndRX Ciphers and Efficient Distinguishers for ARX Designs. *IACR Transactions on Symmetric Cryptology*, 2024(3):84–176, 2024. doi:10.46586/TOSC.V2024.I3.84-176.

[CKY07] Debra L. Cook, Angelos D. Keromytis, and Moti Yung. Elastic block ciphers: the basic design. In Feng Bao and Steven Miller, editors, *Proceedings of the 2007 ACM Symposium on Information, Computer and Communications Security, ASIACCS 2007, Singapore, March 20-22, 2007*, pages 350–352. ACM, 2007. doi:10.1145/1229285.1229324.

[CLH+23] Huiqin Chen, Yongqiang Li, Xichao Hu, Zhengbin Liu, Lin Jiao, and Mingsheng Wang. Automatic Search Model for Related-Tweakey Impossible Differential Cryptanalysis. In Jianying Zhou, Lejla Batina, Zengpeng Li, Jingqiang Lin, Eleonora Losiouk, Suryadipta Majumdar, Daisuke Mashima, Weizhi Meng, Stjepan Picek, Mohammad Ashiqur Rahman, Jun Shao, Masaki Shimaoka, Ezekiel O. Soremekun, Chunhua Su, Je Sen Teh, Aleksei Udovenko, Cong Wang, Leo Yu Zhang, and Yury Zhauniarovich, editors, *Applied Cryptography and Network Security Workshops - ACNS 2023 Satellite Workshops, ADSC, AIBlock, AIHWS, AIoTS, CIMSS, Cloud S&P, SCI, SecMT, SiMLA, Kyoto, Japan, June 19-22, 2023, Proceedings*, volume 13907 of *Lecture Notes in Computer Science*, pages 3–22. Springer, 2023. doi:10.1007/978-3-031-41181-6_1.

[CLMP17] Yu Long Chen, Atul Luykx, Bart Mennink, and Bart Preneel. Efficient Length Doubling From Tweakable Block Ciphers. *IACR Transactions on Symmetric Cryptology*, 2017(3):253–270, 2017. doi:10.13154/TOSC.V2017.I3.253-270.

[CLNY06] Donghoon Chang, Sangjin Lee, Mridul Nandi, and Moti Yung. Indifferentiable Security Analysis of Popular Hash Functions with Prefix-Free Padding. In Xuejia Lai and Kefei Chen, editors, *Advances in Cryptology — ASIACRYPT 2006, 12th International Conference on the Theory and Application of Cryptology and Information Security, Shanghai, China, December 3-7, 2006, Proceedings*, volume 4284 of *Lecture Notes in Computer Science*, pages 283–298. Springer, 2006. doi:10.1007/11935230_19.

[CN08] Donghoon Chang and Mridul Nandi. Improved Indifferentiability Security Analysis of chopMD Hash Function. In Kaisa Nyberg, editor, *Fast Software Encryption, 15th International Workshop, FSE 2008, Lausanne, Switzerland, February 10-13, 2008, Revised Selected Papers*, volume 5086 of *Lecture Notes in Computer Science*, pages 429–443. Springer, 2008. doi:10.1007/978-3-540-71039-4_27.

[Cro00] Paul Crowley. Mercy: A Fast Large Block Cipher for Disk Sector Encryption. In Bruce Schneier, editor, *Fast Software Encryption, 7th International*

Workshop, FSE 2000, New York, NY, USA, April 10-12, 2000, Proceedings, volume 1978 of *Lecture Notes in Computer Science*, pages 49–63. Springer, 2000. doi:10.1007/3-540-44706-7_4.

[CSQ07] Baudoin Collard, François-Xavier Standaert, and Jean-Jacques Quisquater. Improving the Time Complexity of Matsui's Linear Cryptanalysis. In Kil-Hyun Nam and Gwangsoo Rhee, editors, *Information Security and Cryptology - ICISC 2007, 10th International Conference, Seoul, Korea, November 29-30, 2007, Proceedings*, volume 4817 of *Lecture Notes in Computer Science*, pages 77–88. Springer, 2007. doi:10.1007/978-3-540-76788-6_7.

[CXTQ23] Yaxin Cui, Hong Xu, Lin Tan, and Wenfeng Qi. SAT-Aided Differential Cryptanalysis of Lightweight Block Ciphers Midori, MANTIS and QARMA. In Ding Wang, Moti Yung, Zheli Liu, and Xiaofeng Chen, editors, *Information and Communications Security - 25th International Conference, ICICS 2023, Tianjin, China, November 18-20, 2023, Proceedings*, volume 14252 of *Lecture Notes in Computer Science*, pages 3–18. Springer, 2023. doi:10.1007/978-981-99-7356-9_1.

[CYK04] Debra L. Cook, Moti Yung, and Angelos D. Keromytis. Elastic AES. *IACR Cryptology ePrint Archive*, 2004. Available from: http://eprint.iacr.org/2004/141.

[Dam89] Ivan Damgård. A Design Principle for Hash Functions. In Gilles Brassard, editor, *Advances in Cryptology — CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 416–427. Springer, 1989. doi:10.1007/0-387-34805-0_39.

[DBN+01] Morris J. Dworkin, Elaine Barker, James Nechvatal, James Foti, Lawrence E. Bassham, E. Roback, and James Dray, Jr. Advanced Encryption Standard (AES). *Federal information processing standards (FIPS),* National Institute of Standards and Technology, Gaithersburg, MD, November 2001. doi:https://doi.org/10.6028/NIST.FIPS.197.

[DDH+20] Stéphanie Delaune, Patrick Derbez, Paul Huynh, Marine Minier, Victor Mollimard, and Charles Prud'homme. SKINNY with Scalpel - Comparing Tools for Differential Analysis. *IACR Cryptology ePrint Archive*, 2020. Available from: https://eprint.iacr.org/2020/1402.

[DF13] Patrick Derbez and Pierre-Alain Fouque. Exhausting Demirci-Selçuk Meet-in-the-Middle Attacks Against Reduced-Round AES. In Shiho Moriai, editor, *Fast Software Encryption — 20th International Workshop, FSE 2013, Singapore, March 11-13, 2013. Revised Selected Papers*, volume 8424 of *LNCS*, pages 541–560. Springer, 2013. doi:10.1007/978-3-662-43933-3_28.

[DFJ13] Patrick Derbez, Pierre-Alain Fouque, and Jérémy Jean. Improved Key Recovery Attacks on Reduced-Round AES in the Single-Key Setting. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology — EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, volume 7881 of *LNCS*, pages 371–387. Springer, 2013. doi:10.1007/978-3-642-38348-9_23.

[DFJL18] Patrick Derbez, Pierre-Alain Fouque, Jérémy Jean, and Baptiste Lambin. Variants of the AES Key Schedule for Better Truncated Differential

Bounds. In Carlos Cid and Michael J. Jacobson Jr., editors, *Selected Areas in Cryptography - SAC 2018, 25th International Conference, Calgary, AB, Canada, August 15-17, 2018, Revised Selected Papers*, volume 11349 of *Lecture Notes in Computer Science*, pages 27–49. Springer, 2018. doi:10.1007/978-3-030-10970-7_2.

[DFUB24] Hieu Nguyen Duy, Pablo García Fernández, Aleksei Udovenko, and Alex Biryukov. Accordion mode based on Hash-Encrypt-Hash, June 2024. Available from: https://csrc.nist.gov/csrc/media/Events/2024/accordion-cipher-mode-workshop-2024/documents/papers/accordion-mode-based-hash-encrypt-hash.pdf.

[DGK+24] Orr Dunkelman, Shibam Ghosh, Nathan Keller, Gaëtan Leurent, Avichai Marmor, and Victor Mollimard. Partial Sums Meet FFT: Improved Attack on 6-Round AES. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology – EUROCRYPT 2024 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 2630, 2024, Proceedings, Part I*, volume 14651 of *Lecture Notes in Computer Science*, pages 128–157. Springer, 2024. doi:10.1007/978-3-031-58716-0_5.

[DKLS20] Orr Dunkelman, Nathan Keller, Noam Lasry, and Adi Shamir. New Slide Attacks on Almost Self-similar Ciphers. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology — EUROCRYPT 2020. 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 250–279. Springer, 2020. doi:10.1007/978-3-030-45721-1_10.

[DKR97] Joan Daemen, Lars R. Knudsen, and Vincent Rijmen. The Block Cipher Square. In Eli Biham, editor, *Fast Software Encryption, 4th International Workshop, FSE '97, Haifa, Israel, January 20-22, 1997, Proceedings*, volume 1267 of *Lecture Notes in Computer Science*, pages 149–165. Springer, 1997. doi:10.1007/BFB0052343.

[DKRS20] Orr Dunkelman, Nathan Keller, Eyal Ronen, and Adi Shamir. The Retracing Boomerang Attack. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 1014, 2020, Proceedings, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 280–309. Springer, 2020. doi:10.1007/978-3-030-45721-1_11.

[DKRS24] Orr Dunkelman, Nathan Keller, Eyal Ronen, and Adi Shamir. The Retracing Boomerang Attack, with Application to Reduced-Round AES. *Journal of Cryptology*, 37(3):32, 2024. doi:10.1007/S00145-024-09512-7.

[DKS15a] Orr Dunkelman, Nathan Keller, and Adi Shamir. Improved Single-Key Attacks on 8-Round AES-192 and AES-256. *Journal of Cryptology*, 28(3):397–422, 2015. doi:10.1007/s00145-013-9159-4.

[DKS15b] Orr Dunkelman, Nathan Keller, and Adi Shamir. Slidex Attacks on the Even-Mansour Encryption Scheme. *Journal of Cryptology*, 28(1):1–28, 2015. doi:10.1007/S00145-013-9164-7.

[DLP+09] Joan Daemen, Mario Lamberger, Norbert Pramstaller, Vincent Rijmen, and Frederik Vercauteren. Computational aspects of the expected differential probability of 4-round AES and AES-like ciphers. *Computing*, 85(1-2):85–104, 2009. doi:10.1007/S00607-009-0034-Y.

[DLP23] Xiaoyang Dong, Shun Li, and Phuong Pham. Chosen-Key Distinguishing Attacks on Full AES-192, AES-256, Kiasu-BC, and More. *IACR Cryptology ePrint Archive*, 2023. Available from: https://eprint.iacr.org/2023/1095.

[DMMT24] Christoph Dobraunig, Krystian Matusiewicz, Bart Mennink, and Alexander Tereschenko. Efficient Instances of Docked Double Decker With AES. *IACR Cryptology ePrint Archive*, 2024. Available from: https://eprint.iacr.org/2024/084.

[DMN23] Christoph Dobraunig, Bart Mennink, and Samuel Neves. EliMAC: Speeding Up LightMAC by around 20%. *IACR Trans. Symmetric Cryptol.*, 2023(2):69–93, 2023. Available from: https://doi.org/10.46586/tosc.v2023.i2.69-93, doi:10.46586/TOSC.V2023.I2.69-93.

[DR02] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES — The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002. doi:10.1007/978-3-662-04722-4.

[DR05a] Joan Daemen and Vincent Rijmen. A New MAC Construction ALRED and a Specific Instance ALPHA-MAC. In Henri Gilbert and Helena Handschuh, editors, *Fast Software Encryption: 12th International Workshop, FSE 2005, Paris, France, February 21-23, 2005, Revised Selected Papers*, volume 3557 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2005. doi:10.1007/11502760_1.

[DR05b] Joan Daemen and Vincent Rijmen. The Pelican MAC Function. *IACR Cryptol. ePrint Arch.*, 2005. Available from: http://eprint.iacr.org/2005/088.

[DR10] Joan Daemen and Vincent Rijmen. Refinements of the ALRED construction and MAC security claims. *IET Inf. Secur.*, 4(3):149–157, 2010. Available from: https://doi.org/10.1049/iet-ifs.2010.0015, doi:10.1049/IET-IFS.2010.0015.

[DS08] Hüseyin Demirci and Ali Aydin Selçuk. A Meet-in-the-Middle Attack on 8-Round AES. In Kaisa Nyberg, editor, *Fast Software Encryption, 15th International Workshop, FSE 2008, Lausanne, Switzerland, February 10-13, 2008, Revised Selected Papers*, volume 5086 of *LNCS*, pages 116–126. Springer, 2008. doi:10.1007/978-3-540-71039-4_7.

[EKKT18] Zahra Eskandari, Andreas Brasen Kidmose, Stefan Kölbl, and Tyge Tiessen. Finding Integral Distinguishers with Ease. In Carlos Cid and Michael J. Jacobson Jr., editors, *Selected Areas in Cryptography - SAC 2018, 25th International Conference, Calgary, AB, Canada, August 15-17, 2018, Revised Selected Papers*, volume 11349 of *Lecture Notes in Computer Science*, pages 115–138. Springer, 2018. doi:10.1007/978-3-030-10970-7_6.

[ENP19] Maria Eichlseder, Marcel Nageler, and Robert Primas. Analyzing the Linear Keystream Biases in AEGIS. *IACR Cryptology ePrint Archive*, 2019. Available from: https://eprint.iacr.org/2019/1372.

[FGL+24] Antonio Flórez-Gutiérrez, Lorenzo Grassi, Gregor Leander, Ferdinand Sibleyras, and Yosuke Todo. General Practical Cryptanalysis of the Sum of Round-Reduced Block Ciphers and ZIP-AES. In Kai-Min Chung and Yu Sasaki, editors, *Advances in Cryptology — ASIACRYPT 2024, 30th International Conference on the Theory and Application of Cryptology and Information Security, Kolkata, India, December 9-13, 2024, Proceedings, Part IX*, volume 15492 of *Lecture Notes in Computer Science*, pages 280–311. Springer, 2024. doi:10.1007/978-981-96-0947-5_10.

[FJP13] Pierre-Alain Fouque, Jérémy Jean, and Thomas Peyrin. Structural Evaluation of AES and Chosen-Key Distinguisher of 9-Round AES-128. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology — CRYPTO 2013, 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 183–203. Springer, 2013. doi:10.1007/978-3-642-40041-4_11.

[FKL+00] Niels Ferguson, John Kelsey, Stefan Lucks, Bruce Schneier, Michael Stay, David A. Wagner, and Doug Whiting. Improved Cryptanalysis of Rijndael. In Bruce Schneier, editor, *Fast Software Encryption, 7th International Workshop, FSE 2000, New York, NY, USA, April 10-12, 2000, Proceedings*, volume 1978 of *LNCS*, pages 213–230. Springer, 2000. doi:10.1007/3-540-44706-7_15.

[FLS+10] Niels Ferguson, Stefan Lucks, Bruce Schneier, Doug Whiting Hifn, Mihir Bellare, Tadayoshi Kohno, Jon Callas, and Jesse Walker. The Skein Hash Function Family, Version 1.3, October 2010.

[FWG+16] Kai Fu, Meiqin Wang, Yinghua Guo, Siwei Sun, and Lei Hu. MILP-Based Automatic Search Algorithms for Differential and Linear Trails for Speck. In Thomas Peyrin, editor, *Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*, volume 9783 of *Lecture Notes in Computer Science*, pages 268–288. Springer, 2016. doi:10.1007/978-3-662-52993-5_14.

[GC09] Kris Gaj and Pawel Chodowiec. FPGA and ASIC Implementations of AES. In Çetin Kaya Koç, editor, *Cryptographic Engineering*, pages 235–294. Springer, 2009. doi:10.1007/978-0-387-71817-0_10.

[GJNS14] Jian Guo, Jérémy Jean, Ivica Nikolic, and Yu Sasaki. Meet-in-the-Middle Attacks on Generic Feistel Constructions. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology — ASIACRYPT 2014. 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I*, volume 8873 of *Lecture Notes in Computer Science*, pages 458–477. Springer, 2014. doi:10.1007/978-3-662-45611-8_24.

[GM00] Henri Gilbert and Marine Minier. A Collision Attack on 7 Rounds of Rijndael. In *The Third Advanced Encryption Standard Candidate Conference, April 13-14, 2000, New York, New York, USA*, pages 230–241. National Institute of Standards and Technology„ 2000.

[GM16] Shay Gueron and Nicky Mouha. Simpira v2: A Family of Efficient Permutations Using the AES Round Function. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology — ASIACRYPT 2016, 22nd International Conference on the Theory and Application of Cryptology and*

Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I, volume 10031 of *Lecture Notes in Computer Science*, pages 95–125, 2016. doi:10.1007/978-3-662-53887-6_4.

[GMS16] David Gérault, Marine Minier, and Christine Solnon. Constraint Programming Models for Chosen Key Differential Cryptanalysis. In Michel Rueher, editor, *Principles and Practice of Constraint Programming - 22nd International Conference, CP 2016, Toulouse, France, September 5-9, 2016, Proceedings*, volume 9892 of *Lecture Notes in Computer Science*, pages 584–601. Springer, 2016. doi:10.1007/978-3-319-44953-1_37.

[Gra18] Lorenzo Grassi. Mixture Differential Cryptanalysis: a New Approach to Distinguishers and Attacks on round-reduced AES. *IACR Transactions on Symmetric Cryptology*, 2018(2):133–160, 2018. doi:10.13154/TOSC.V2018.I2.133-160.

[Gro97] Lov K. Grover. Quantum Mechanics Helps in Searching for a Needle in a Haystack. *Physical Review Letters*, 79:325–328, Jul 1997. Available from: https://link.aps.org/doi/10.1103/PhysRevLett.79.325, doi:10.1103/PhysRevLett.79.325.

[GRR17] Lorenzo Grassi, Christian Rechberger, and Sondre Rønjom. A New Structural-Differential Property of 5-Round AES. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology — EUROCRYPT 2017, 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 May 4, 2017, Proceedings, Part II*, volume 10211 of *Lecture Notes in Computer Science*, pages 289–317, 2017. doi:10.1007/978-3-319-56614-6_10.

[Gue12] Shay Gueron. White Paper: Intel®Advanced Encryption Standard (Intel®AES) Instructions Set — Rev 3.01, September 2012. Available from: https://www.intel.com/content/www/us/en/developer/articles/tool/intel-advanced-encryption-standard-aes-instructions-set.html.

[HGSE24] Hosein Hadipour, Simon Gerhalter, Sadegh Sadeghi, and Maria Eichlseder. Improved Search for Integral, Impossible-Differential and Zero-Correlation Attacks: Application to Ascon, ForkSKINNY, SKINNY, MANTIS, PRESENT and QARMAv2. *IACR Transactions on Symmetric Cryptology*, 1, 2024. Available from: https://eprint.iacr.org/2023/1701.

[HI19] Akinori Hosoyamada and Tetsu Iwata. 4-Round Luby-Rackoff Construction is a qPRP. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology — ASIACRYPT 2019, 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part I*, volume 11921 of *Lecture Notes in Computer Science*, pages 145–174. Springer, 2019. doi:10.1007/978-3-030-34578-5_6.

[HKR01] Helena Handschuh, Lars R. Knudsen, and Matthew J. B. Robshaw. Analysis of SHA-1 in Encryption Mode. In David Naccache, editor, *Topics in Cryptology — CT-RSA 2001, The Cryptographer's Track at RSA Conference 2001, San Francisco, CA, USA, April 8-12, 2001, Proceedings*, volume 2020 of *Lecture Notes in Computer Science*, pages 70–83. Springer, 2001. doi:10.1007/3-540-45353-9_7.

[HKR15] Viet Tung Hoang, Ted Krovetz, and Phillip Rogaway. Robust Authenticated-Encryption AEZ and the Problem That it Solves. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology — EUROCRYPT 2015, 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 15–44. Springer, 2015. doi:10.1007/978-3-662-46800-5_2.

[HR04] Shai Halevi and Phillip Rogaway. A Parallelizable Enciphering Mode. In Tatsuaki Okamoto, editor, *Topics in Cryptology — CT-RSA 2004, The Cryptographers' Track at the RSA Conference 2004, San Francisco, CA, USA, February 23-27, 2004, Proceedings*, volume 2964 of *Lecture Notes in Computer Science*, pages 292–304. Springer, 2004. doi:10.1007/978-3-540-24660-2_23.

[HS18] Akinori Hosoyamada and Yu Sasaki. Quantum Demiric-Selçuk Meet-in-the-Middle Attacks: Applications to 6-Round Generic Feistel Constructions. In Dario Catalano and Roberto De Prisco, editors, *Security and Cryptography for Networks - 11th International Conference, SCN 2018, Amalfi, Italy, September 5-7, 2018, Proceedings*, volume 11035 of *Lecture Notes in Computer Science*, pages 386–403. Springer, 2018. doi:10.1007/978-3-319-98113-0_21.

[HSE23] Hosein Hadipour, Sadegh Sadeghi, and Maria Eichlseder. Finding the Impossible: Automated Search for Full Impossible-Differential, Zero-Correlation, and Integral Attacks. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology — EUROCRYPT 2023, 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part IV*, volume 14007 of *Lecture Notes in Computer Science*, pages 128–157. Springer, 2023. doi:10.1007/978-3-031-30634-1_5.

[HSH+08] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. Lest We Remember: Cold Boot Attacks on Encryption Keys. In Paul C. van Oorschot, editor, *Proceedings of the 17th USENIX Security Symposium, July 28-August 1, 2008, San Jose, CA, USA*, pages 45–60. USENIX Association, 2008. Available from: http://www.usenix.org/events/sec08/tech/full_papers/halderman/halderman.pdf.

[HSH+09] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. Lest we remember: cold-boot attacks on encryption keys. *Commun. ACM*, 52(5):91–98, 2009. doi:10.1145/1506409.1506429.

[IBM18] IBM. Power ISA. Technical Report 2.07B, January 2018. Available from: https://ibm.ent.box.com/s/jd5w15gz301s5b5dt375mshpq9c3lh4u.

[IHM+19] Gembu Ito, Akinori Hosoyamada, Ryutaroh Matsumoto, Yu Sasaki, and Tetsu Iwata. Quantum Chosen-Ciphertext Attacks Against Feistel Ciphers. In Mitsuru Matsui, editor, *Topics in Cryptology — CT-RSA 2019, The Cryptographers' Track at the RSA Conference 2019, San Francisco, CA, USA, March 4-8, 2019, Proceedings*, volume 11405 of *Lecture Notes in Computer Science*, pages 391–411. Springer, 2019. doi:10.1007/978-3-030-12612-4_20.

[IMV16] Tetsu Iwata, Bart Mennink, and Damian Vizár. CENC is Optimally Secure. *IACR Cryptology ePrint Archive*, 2016. Available from: http://eprint.iacr.org/2016/1087.

[Int23a] Intel. Intel 64 and IA-32 Architectures Software Developers Manual Volume 1: Basic Architecture, 2023. Available from: http://software.intel.com/en-us/articles/intel-sdm.

[Int23b] Intel. Intel 64 and IA-32 Architectures Software Developers Manual Volume 2: Instruction Set Reference A-Z, 2023. Available from: http://software.intel.com/en-us/articles/intel-sdm.

[IS22] Murat Burhan Ilter and Ali Aydin Selçuk. MILP-Aided Cryptanalysis of the FUTURE Block Cipher. In Giampaolo Bella, Mihai Doinea, and Helge Janicke, editors, *Innovative Security Solutions for Information Technology and Communications - 15th International Conference, SecITC 2022, Virtual Event, December 8-9, 2022, Revised Selected Papers*, volume 13809 of *Lecture Notes in Computer Science*, pages 153–167. Springer, 2022. doi:10.1007/978-3-031-32636-3_9.

[Iwa06] Tetsu Iwata. New Blockcipher Modes of Operation with Beyond the Birthday Bound Security. In Matthew J. B. Robshaw, editor, *Fast Software Encryption, 13th International Workshop, FSE 2006, Graz, Austria, March 15-17, 2006, Revised Selected Papers*, volume 4047 of *Lecture Notes in Computer Science*, pages 310–327. Springer, 2006. doi:10.1007/11799313_20.

[JNP14] Jérémy Jean, Ivica Nikolic, and Thomas Peyrin. Tweaks and Keys for Block Ciphers: The TWEAKEY Framework. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology — ASIACRYPT 2014, 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014, Proceedings, Part II*, volume 8874 of *LNCS*, pages 274–288. Springer, 2014. doi:10.1007/978-3-662-45608-8_15.

[Joh23] Dougall Johnson. Apple Microarchitecture Research: Firestorm and Icestorm, 2021–2023. Available from: https://dougallj.github.io/applecpu/firestorm.html.

[Jr.78] Sheldon B. Akers Jr. Binary Decision Diagrams. *IEEE Trans. Computers*, 27(6):509–516, 1978. doi:10.1109/TC.1978.1675141.

[KLMR16] Stefan Kölbl, Martin M. Lauridsen, Florian Mendel, and Christian Rechberger. Haraka v2 - Efficient Short-Input Hashing for Post-Quantum Applications. *IACR Transactions on Symmetric Cryptology*, 2016(2):1–29, 2016. doi:10.13154/TOSC.V2016.I2.1-29.

[KLPS17] Khoongming Khoo, Eugene Lee, Thomas Peyrin, and Siang Meng Sim. Human-readable Proof of the Related-Key Security of AES-128. *IACR Transactions on Symmetric Cryptology*, 2017(2):59–83, 2017. doi:10.13154/TOSC.V2017.I2.59-83.

[KLT15] Stefan Kölbl, Gregor Leander, and Tyge Tiessen. Observations on the SIMON Block Cipher Family. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology — CRYPTO 2015, 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, volume 9215 of *LNCS*, pages 161–185. Springer, 2015. doi:10.1007/978-3-662-47989-6_8.

[KLW17] Thorsten Kranz, Gregor Leander, and Friedrich Wiemer. Linear Cryptanalysis: Key Schedules and Tweakable Block Ciphers. *IACR Transactions on Symmetric Cryptology*, 2017(1):474–505, 2017. doi:10.13154/TOSC.V2017.I1.474-505.

[Knu98] Lars Knudsen. DEAL–A 128-bit Block Cipher. Technical report, Department of Informatics, University of Bergen, Norway, 1998.

[KS07] Liam Keliher and Jiayuan Sui. Exact maximum expected differential and linear probability for two-round Advanced Encryption Standard. *IET Inf. Secur.*, 1(2):53–57, 2007. Available from: https://doi.org/10.1049/iet-ifs:20060161, doi:10.1049/IET-IFS:20060161.

[KY10] Abdel Alim Kamal and Amr M. Youssef. Applications of SAT Solvers to AES Key Recovery from Decayed Key Schedule Images. In Reijo Savola, Masaru Takesue, Rainer Falk, and Manuela Popescu, editors, *Fourth International Conference on Emerging Security Information Systems and Technologies, SECURWARE 2010, Venice, Italy, July 18-25, 2010*, pages 216–220. IEEE Computer Society, 2010. doi:10.1109/SECURWARE.2010.42.

[Lem21] Daniel Lemire. Counting cycles and instructions on the Apple M1 processor, March 2021. Available from: https://lemire.me/blog/2021/03/24/.

[Lem23] Daniel Lemire. Counting cycles and instructions on Apple M-series systems, March 2023. Available from: https://lemire.me/blog/2023/03/21/.

[LKK16] Anatoly Lebedev, Andrey Karondeev, and Alexandre Kozlov. New block Cipher — Eurocrypt 2016 Rump Session Presentation, 2016. Available from: https://www.iacr.org/conferences/eurocrypt2016/slides/121.pdf.

[LP21] Gaëtan Leurent and Clara Pernot. New Representations of the AES Key Schedule. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021, 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 1721, 2021, Proceedings, Part I*, volume 12696 of *Lecture Notes in Computer Science*, pages 54–84. Springer, 2021. doi:10.1007/978-3-030-77870-5_3.

[LP25] Gaëtan Leurent and Clara Pernot. New Representations of the AES Key Schedule. *Journal of Cryptology*, 38(1):2, 2025. doi:10.1007/S00145-024-09522-5.

[LR88] Michael Luby and Charles Rackoff. How to Construct Pseudorandom Permutations from Pseudorandom Functions. *SIAM J. Comput.*, 17(2):373–386, 1988. doi:10.1137/0217022.

[LW22] Yingying Li and Qichun Wang. The SAT-Based Automatic Searching and Experimental Verification for Differential Characteristics with Application to Midori64. In Willy Susilo, Xiaofeng Chen, Fuchun Guo, Yudi Zhang, and Rolly Intan, editors, *Information Security - 25th International Conference, ISC 2022, Bali, Indonesia, December 18-22, 2022, Proceedings*, volume 13640 of *Lecture Notes in Computer Science*, pages 153–161. Springer, 2022. doi:10.1007/978-3-031-22390-7_10.

[Mat93] Mitsuru Matsui. Linear Cryptanalysis Method for DES Cipher. In Tor Helleseth, editor, *Advances in Cryptology — EUROCRYPT '93, Lofthus, Norway, May 23-27, 1993, Proceedings*, volume 765 of *Lecture Notes in Computer Science*, pages 386–397. Springer, 1993. doi:10.1007/3-540-48285-7_33.

[Max19] Alexander Maximov. AES MixColumn with 92 XOR gates. *IACR Cryptology ePrint Archive*, 2019. Available from: https://eprint.iacr.org/2019/833.

[MDRM10] Hamid Mala, Mohammad Dakhilalian, Vincent Rijmen, and Mahmoud Modarres-Hashemi. Improved Impossible Differential Cryptanalysis of 7-Round AES-128. In Guang Gong and Kishan Chand Gupta, editors, *Progress in Cryptology — INDOCRYPT 2010, 11th International Conference on Cryptology in India, Hyderabad, India, December 12-15, 2010. Proceedings*, volume 6498 of *LNCS*, pages 282–291. Springer, 2010. doi:10.1007/978-3-642-17401-8_20.

[ME19] Alexander Maximov and Patrik Ekdahl. New Circuit Minimization Techniques for Smaller and Faster AES SBoxes. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(4):91–125, 2019. Available from: https://doi.org/10.13154/tches.v2019.i4.91-125, doi:10.13154/TCHES.V2019.I4.91-125.

[Men15] Bart Mennink. Optimally Secure Tweakable Blockciphers. Cryptology ePrint Archive, Paper 2015/363, 2015. Available from: https://eprint.iacr.org/2015/363.

[Mer79] Ralph Charles Merkle. *Secrecy, authentication, and public key systems*. PhD thesis, Department of Electrical Engineering, Stanford University, 1979.

[Mer89] Ralph Charles Merkle. One Way Hash Functions and DES. In Gilles Brassard, editor, *Advances in Cryptology — CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 428–446. Springer, 1989. doi:10.1007/0-387-34805-0_40.

[MMO85] Stephen M. Matyas, Carl H. Meyer, and Jonathan Oseas. Generating strong one-way functions with cryptographic algorithms. *IBM Technical Disclosure Bulletin*, 27:5658–5659, 1985.

[MN17] Bart Mennink and Samuel Neves. Optimal PRFs from Blockcipher Designs. *IACR Transactions on Symmetric Cryptology*, 2017(3):228–252, 2017. doi:10.13154/TOSC.V2017.I3.228-252.

[MNP+21] Ben Marshall, G. Richard Newell, Dan Page, Markku-Juhani O. Saarinen, and Claire Wolf. The design of scalar AES Instruction Set Extensions for RISC-V. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(1):109–136, 2021. doi:10.46586/TCHES.V2021.I1.109-136.

[MNSV97] David M'Raïhi, David Naccache, Jacques Stern, and Serge Vaudenay. XMX: A Firmware-Oriented Block Cipher Based on Modular Multiplications. In *Fast Software Encryption, 4th International Workshop, FSE '97, Haifa, Israel, January 20-22, 1997, Proceedings*, volume 1267 of *Lecture Notes in Computer Science*, pages 166–171. Springer, 1997. doi:10.1007/BFb0052344.

[MP13] Nicky Mouha and Bart Preneel. Towards Finding Optimal Differential Characteristics for ARX: Application to Salsa20. Cryptology ePrint Archive, Paper 2013/328, 2013. Available from: https://eprint.iacr.org/2013/328.

[MRSA23] Sandip Kumar Mondal, Mostafizar Rahman, Santanu Sarkar, and Avishek Adhikari. Revisiting Yoyo Tricks on AES. *IACR Trans. Symmetric Cryptol.*, 2023(4):28–57, 2023. Available from: https://doi.org/10.46586/tosc.v2023.i4.28-57, doi:10.46586/TOSC.V2023.I4.28-57.

[MS02] Sumio Morioka and Akashi Satoh. A 10 Gbps Full-AES Crypto Design with a Twisted-BDD S-Box Architecture. In *20th International Conference on Computer Design (ICCD 2002), VLSI in Computers and Processors, 16-18 September 2002, Freiburg, Germany, Proceedings*, pages 98–103. IEEE Computer Society, 2002. doi:10.1109/ICCD.2002.1106754.

[MT06] Kazuhiko Minematsu and Yukiyasu Tsunoo. Provably Secure MACs from Differentially-Uniform Permutations and AES-Based Implementations. In Matthew J. B. Robshaw, editor, *Fast Software Encryption, 13th International Workshop, FSE 2006, Graz, Austria, March 15-17, 2006, Revised Selected Papers*, volume 4047 of *Lecture Notes in Computer Science*, pages 226–241. Springer, 2006. doi:10.1007/11799313_15.

[MWGP11] Nicky Mouha, Qingju Wang, Dawu Gu, and Bart Preneel. Differential and Linear Cryptanalysis Using Mixed-Integer Linear Programming. In Chuankun Wu, Moti Yung, and Dongdai Lin, editors, *Information Security and Cryptology — 7th International Conference, Inscrypt 2011, Beijing, China, November 30 - December 3, 2011. Revised Selected Papers*, volume 7537 of *Lecture Notes in Computer Science*, pages 57–76. Springer, 2011. doi:10.1007/978-3-642-34704-7_5.

[NR95] Moni Naor and Omer Reingold. Synthesizers and Their Application to the Parallel Construction of Pseudo-random Functions. *Electron. Colloquium Comput. Complex.*, TR95-045, 1995. Available from: https://eccc.weizmann.ac.il/eccc-reports/1995/TR95-045/index.html, arXiv:TR95-045.

[NR98] Moni Naor and Omer Reingold. From Unpredictability to Indistinguishability: A Simple Construction of Pseudo-Random Functions from MACs (Extended Abstract). In Hugo Krawczyk, editor, *Advances in Cryptology - CRYPTO '98, 18th Annual International Cryptology Conference, Santa Barbara, California, USA, August 23-27, 1998, Proceedings*, volume 1462 of *Lecture Notes in Computer Science*, pages 267–282. Springer, 1998. Available from: https://doi.org/10.1007/BFb0055734, doi:10.1007/BFB0055734.

[NR99a] Moni Naor and Omer Reingold. On the Construction of Pseudorandom Permutations: Luby-Rackoff Revisited. *Journal of Cryptology*, 12(1):29–66, 1999. doi:10.1007/PL00003817.

[NR99b] Moni Naor and Omer Reingold. A Pseudo-Random Encryption Mode, 1999. Available from: https://omereingold.wordpress.com/wp-content/uploads/2014/10/nr_mode.pdf.

[NSA+23] Motoki Nakahashi, Rentaro Shiba, Ravi Anand, Mostafizar Rahman, Kosei Sakamoto, Fukang Liu, and Takanori Isobe. Ghidle: Efficient Large-State

Block Ciphers for Post-quantum Security. In Leonie Simpson and Mir Ali Rezazadeh Baee, editors, *Information Security and Privacy - 28th Australasian Conference, ACISP 2023, Brisbane, QLD, Australia, July 5-7, 2023, Proceedings*, volume 13915 of *Lecture Notes in Computer Science*, pages 403–430. Springer, 2023. doi:10.1007/978-3-031-35486-1_18.

[OGK+15] Roman Oliynykov, Ivan Gorbenko, Oleksandr Kazymyrov, Victor Ruzhentsev, Oleksandr Kuznetsov, Yurii Gorbenko, Oleksandr Dyrda, Viktor Dolgov, Andrii Pushkaryov, Ruslan Mordvinov, and Dmytro Kaidalov. A New Encryption Standard of Ukraine: The Kalyna Block Cipher. *IACR Cryptology ePrint Archive*, 2015. Available from: http://eprint.iacr.org/2015/650.

[Pie90] Josef Pieprzyk. How to Construct Pseudorandom Permutations from Single Pseudorandom Functions. In Ivan Damgård, editor, *Advances in Cryptology — EUROCRYPT '90, Aarhus, Denmark, May 21-24, 1990, Proceedings*, volume 473 of *Lecture Notes in Computer Science*, pages 140–150. Springer, 1990. doi:10.1007/3-540-46877-3_12.

[Pre01] Bart Preneel. NESSIE: A European Approach to Evaluate Cryptographic Algorithms. In Mitsuru Matsui, editor, *Fast Software Encryption, 8th International Workshop, FSE 2001 Yokohama, Japan, April 2-4, 2001, Revised Papers*, volume 2355 of *Lecture Notes in Computer Science*, pages 267–276. Springer, 2001. doi:10.1007/3-540-45473-X_22.

[PS16] Thomas Peyrin and Yannick Seurin. Counter-in-Tweak: Authenticated Encryption Modes for Tweakable Block Ciphers. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology — CRYPTO 2016, 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I*, volume 9814 of *LNCS*, pages 33–63. Springer, 2016. doi:10.1007/978-3-662-53018-4_2.

[QCW16] Lingyue Qin, Huaifeng Chen, and Xiaoyun Wang. Linear Hull Attack on Round-Reduced Simeck with Dynamic Key-Guessing Techniques. In Joseph K. Liu and Ron Steinfeld, editors, *Information Security and Privacy - 21st Australasian Conference, ACISP 2016, Melbourne, VIC, Australia, July 4-6, 2016, Proceedings, Part II*, volume 9723 of *Lecture Notes in Computer Science*, pages 409–424. Springer, 2016. doi:10.1007/978-3-319-40367-0_26.

[QDW+21] Lingyue Qin, Xiaoyang Dong, Xiaoyun Wang, Keting Jia, and Yunwen Liu. Automated Search Oriented to Key Recovery on Ciphers with Linear Key Schedule Applications to Boomerangs in SKINNY and ForkSkinny. *IACR Transactions on Symmetric Cryptology*, 2021(2):249–291, 2021. doi:10.46586/TOSC.V2021.I2.249-291.

[RBH17] Sondre Rønjom, Navid Ghaedi Bardeh, and Tor Helleseth. Yoyo Tricks with AES. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT I*, volume 10624 of *Lecture Notes in Computer Science*, pages 217–243. Springer, 2017. doi:10.1007/978-3-319-70694-8_8.

[RR07] Thomas Ristenpart and Phillip Rogaway. How to Enrich the Message Space of a Cipher. In Alex Biryukov, editor, *Fast Software Encryption, 14th International Workshop, FSE 2007, Luxembourg, Luxembourg, March 26-28, 2007, Revised Selected Papers*, volume 4593 of *Lecture Notes in Computer Science*, pages 101–118. Springer, 2007. doi:10.1007/978-3-540-74619-5_7.

[RS06] Phillip Rogaway and Thomas Shrimpton. A Provable-Security Treatment of the Key-Wrap Problem. In Serge Vaudenay, editor, *Advances in Cryptology — EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings*, volume 4004 of *Lecture Notes in Computer Science*, pages 373–390. Springer, 2006. doi:10.1007/11761679_23.

[RS09] Mathieu Renauld and François-Xavier Standaert. Algebraic Side-Channel Attacks. In Feng Bao, Moti Yung, Dongdai Lin, and Jiwu Jing, editors, *Information Security and Cryptology — 5th International Conference, Inscrypt 2009, Beijing, China, December 12-15, 2009. Revised Selected Papers*, volume 6151 of *Lecture Notes in Computer Science*, pages 393–410. Springer, 2009. doi:10.1007/978-3-642-16342-5_29.

[RSP21] Mostafizar Rahman, Dhiman Saha, and Goutam Paul. Boomeyong: Embedding Yoyo within Boomerang and its Applications to Key Recovery Attacks on AES and Pholkos. *IACR Transactions on Symmetric Cryptology*, 2021(3):137–169, 2021. doi:10.46586/TOSC.V2021.I3.137-169.

[RZBM24] Mahnaz Namazi Rizi, Nusa Zidaric, Lejla Batina, and Nele Mentens. Optimised AES with RISC-V Vector Extensions. In *27th International Symposium on Design & Diagnostics of Electronic Circuits & Systems, DDECS 2024, Kielce, Poland, April 3-5, 2024*, pages 57–60. IEEE, 2024. doi:10.1109/DDECS60919.2024.10508919.

[Sas93] Tsutomu Sasao. AND-EXOR expressions and their optimization. In *Logic synthesis and optimization*, pages 287–312. Springer, 1993.

[SGL+17] Siwei Sun, David Gérault, Pascal Lafourcade, Qianqian Yang, Yosuke Todo, Kexin Qiao, and Lei Hu. Analysis of AES, SKINNY, and Others with Constraint Programming. *IACR Transactions on Symmetric Cryptology*, 2017(1):281–306, 2017. doi:10.13154/TOSC.V2017.I1.281-306.

[SHW+14] Siwei Sun, Lei Hu, Peng Wang, Kexin Qiao, Xiaoshuang Ma, and Ling Song. Automatic Security Evaluation and (Related-key) Differential Characteristic Search: Application to SIMON, PRESENT, LBlock, DES(L) and Other Bit-Oriented Block Ciphers. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology — ASIACRYPT 2014, 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I*, volume 8873 of *Lecture Notes in Computer Science*, pages 158–178. Springer, 2014. doi:10.1007/978-3-662-45611-8_9.

[SHY16] Ling Song, Zhangjie Huang, and Qianqian Yang. Automatic Differential Analysis of ARX Block Ciphers with Application to SPECK and LEA. In Joseph K. Liu and Ron Steinfeld, editors, *Information Security and Privacy — 21st Australasian Conference, ACISP 2016, Melbourne, VIC, Australia, July 4-6, 2016, Proceedings, Part II*, volume 9723 of *LNCS*, pages 379–394. Springer, 2016. doi:10.1007/978-3-319-40367-0_24.

[SII23] Kosei Sakamoto, Ryoma Ito, and Takanori Isobe. Parallel SAT Framework to Find Clustering of Differential Characteristics and Its Applications. In Claude Carlet, Kalikinkar Mandal, and Vincent Rijmen, editors, *Selected Areas in Cryptography - SAC 2023, 30th International Conference, Fredericton, Canada, August 14-18, 2023, Revised Selected Papers*, volume

14201 of *Lecture Notes in Computer Science*, pages 409–428. Springer, 2023. doi:10.1007/978-3-031-53368-6_20.

[SLG+16] Bing Sun, Meicheng Liu, Jian Guo, Vincent Rijmen, and Ruilin Li. Provable Security Evaluation of Structures Against Impossible Differential and Zero Correlation Linear Cryptanalysis. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part I*, volume 9665 of *Lecture Notes in Computer Science*, pages 196–213. Springer, 2016. doi:10.1007/978-3-662-49890-3_8.

[SLR+15] Bing Sun, Zhiqiang Liu, Vincent Rijmen, Ruilin Li, Lei Cheng, Qingju Wang, Hoda Alkhzaimi, and Chao Li. Links Among Impossible Differential, Integral and Zero Correlation Linear Cryptanalysis. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 95–115. Springer, 2015. doi:10.1007/978-3-662-47989-6_5.

[SRB18] Sukanya Saha, Krishnendu Rarhi, and Abhishek Bhattacharya. Systematization Of A 256-Bit Lightweight Block Cipher Marvin. *IACR Cryptology ePrint Archive*, 2018. Available from: http://eprint.iacr.org/2018/064.

[SSD+18] Danping Shi, Siwei Sun, Patrick Derbez, Yosuke Todo, Bing Sun, and Lei Hu. Programming the Demirci-Selçuk Meet-in-the-Middle Attack with Constraints. In Thomas Peyrin and Steven D. Galbraith, editors, *Advances in Cryptology — ASIACRYPT 2018, 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part II*, volume 11273 of *Lecture Notes in Computer Science*, pages 3–34. Springer, 2018. doi:10.1007/978-3-030-03329-3_1.

[SSI22] Rentaro Shiba, Kosei Sakamoto, and Takanori Isobe. Efficient constructions for large-state block ciphers based on AES New Instructions. *IET Inf. Secur.*, 16(3):145–160, 2022. doi:10.1049/ISE2.12053.

[STGD12] Ion Sima, Daniel Tarmurean, Victor Greu, and Adrian-Viorel Diaconu. XXTEA, an alternative replacement of KASUMI cipher algorithm in A5/3 GSM and f8, f9 UMTS data security functions. In *9th International Conference on Communications, COMM 2012, Bucharest, Romania, June 21-23, 2012*, pages 323–326. IEEE, 2012. doi:10.1109/ICCOMM.2012.6262617.

[Sun21] Bing Sun. Provable Security Evaluation of Block Ciphers Against Demirci-Selçuk's Meet-in-the-Middle Attack. *IEEE Trans. Inf. Theory*, 67(7):4838–4844, 2021. doi:10.1109/TIT.2021.3058377.

[SWW21] Ling Sun, Wei Wang, and Meiqin Wang. Accelerating the Search of Differential and Linear Characteristics with the SAT Method. *IACR Transactions on Symmetric Cryptology*, 2021(1):269–315, 2021. doi:10.46586/TOSC.V2021.I1.269-315.

[TA14] Yosuke Todo and Kazumaro Aoki. FFT Key Recovery for Integral Attack. In Dimitris Gritzalis, Aggelos Kiayias, and Ioannis G. Askoxylakis, editors, *CANS*, volume 8813 of *Lecture Notes in Computer Science*, pages 64–81. Springer, 2014. doi:10.1007/978-3-319-12280-9_5.

[TSS+24] Atsushi Tanaka, Rentaro Shiba, Kosei Sakamoto, Mostafizar Rahman, Takuro Shiraya, and Takanori Isobe. ASURA: An Efficient Large-State Tweakable Block Cipher for ARM Environment. In Sourav Mukhopadhyay and Pantelimon Stanica, editors, *Progress in Cryptology — INDOCRYPT 2024, 25th International Conference on Cryptology in India, Chennai, India, December 18-21, 2024, Proceedings, Part I*, volume 15495 of *Lecture Notes in Computer Science*, pages 143–164. Springer, 2024. doi:10.1007/978-3-031-80308-6_7.

[Wag99] David A. Wagner. The Boomerang Attack. In Lars R. Knudsen, editor, *Fast Software Encryption, 6th International Workshop, FSE '99, Rome, Italy, March 24-26, 1999, Proceedings*, volume 1636 of *Lecture Notes in Computer Science*, pages 156–170. Springer, 1999. doi:10.1007/3-540-48519-8_12.

[WCJ+21] Yoo-Seung Won, Soham Chatterjee, Dirmanto Jap, Arindam Basu, and Shivam Bhasin. DeepFreeze: Cold Boot Attacks and High Fidelity Model Recovery on Commercial EdgeML Device. In *Proceedings of ICCAD 2021*, pages 1–9. IEEE, 2021. doi:10.1109/ICCAD51958.2021.9643512.

[WW11] Shengbao Wu and Mingsheng Wang. Security Evaluation against Differential Cryptanalysis for Block Cipher Structures. *IACR Cryptology ePrint Archive*, 2011. Available from: http://eprint.iacr.org/2011/551.

[WWHL18] Xuzi Wang, Baofeng Wu, Lin Hou, and Dongdai Lin. Automatic Search for Related-Key Differential Trails in SIMON-like Block Ciphers Based on MILP. In Liqun Chen, Mark Manulis, and Steve A. Schneider, editors, *Information Security - 21st International Conference, ISC 2018, Guildford, UK, September 9-12, 2018, Proceedings*, volume 11060 of *Lecture Notes in Computer Science*, pages 116–131. Springer, 2018. doi:10.1007/978-3-319-99136-8_7.

[XY23] Yinsong Xu and Zheng Yuan. Quantum meet-in-the-middle attack on Feistel construction. *Quantum Inf. Process.*, 22(3):155, 2023. doi:10.1007/S11128-022-03715-2.

[XZBL16] Zejun Xiang, Wentao Zhang, Zhenzhen Bao, and Dongdai Lin. Applying MILP Method to Searching Integral Distinguishers Based on Division Property for 6 Lightweight Block Ciphers. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology — ASIACRYPT 2016, 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 648–678, 2016. doi:10.1007/978-3-662-53887-6_24.

[XZH20] Zhaohui Xing, Wenying Zhang, and Guoyong Han. Improved Conditional Differential Analysis on NLFSR-Based Block Cipher KATAN32 with MILP. *Wirel. Commun. Mob. Comput.*, 2020:8883557:1–8883557:14, 2020. doi:10.1155/2020/8883557.

[YADA17] Salessawi Ferede Yitbarek, Misiker Tadesse Aga, Reetuparna Das, and Todd M. Austin. Cold Boot Attacks are Still Hot: Security Analysis of Memory Scramblers in Modern Processors. In *2017 IEEE International Symposium on High Performance Computer Architecture, HPCA 2017, Austin, TX, USA, February 4-8, 2017*, pages 313–324. IEEE Computer Society, 2017. doi:10.1109/HPCA.2017.10.

[YML+17] Jun Yin, Chuyan Ma, Lijun Lyu, Jian Song, Guang Zeng, Chuangui Ma, and Fushan Wei. Improved Cryptanalysis of an ISO Standard Lightweight Block Cipher with Refined MILP Modelling. In Xiaofeng Chen, Dongdai Lin, and Moti Yung, editors, *Information Security and Cryptology — 13th International Conference, Inscrypt 2017, Xi'an, China, November 3-5, 2017, Revised Selected Papers*, volume 10726 of *Lecture Notes in Computer Science*, pages 404–426. Springer, 2017. doi:10.1007/978-3-319-75160-3_24.

[YTC20] Wei-Zhu Yeoh, Je Sen Teh, and Jiageng Chen. Automated Search for Block Cipher Differentials: A GPU-Accelerated Branch-and-Bound Algorithm. In Joseph K. Liu and Hui Cui, editors, *Information Security and Privacy - 25th Australasian Conference, ACISP 2020, Perth, WA, Australia, November 30 - December 2, 2020, Proceedings*, volume 12248 of *Lecture Notes in Computer Science*, pages 160–179. Springer, 2020. doi:10.1007/978-3-030-55304-3_9.

[YTQ23] Xueping Yan, Lin Tan, and Wenfeng Qi. Non-Existence of One-Byte Active Impossible Differentials for 5-Round AES in the Master-Key Setting. *J. Syst. Sci. Complex.*, 36(3):1336–1350, 2023. Available from: https://doi.org/10.1007/s11424-023-1307-9, doi:10.1007/S11424-023-1307-9.

[YTXQ24] Xueping Yan, Lin Tan, Hong Xu, and Wen-Feng Qi. The Boomerang Chain Distinguishers: New Record for 6-Round AES. In Kai-Min Chung and Yu Sasaki, editors, *Advances in Cryptology — ASIACRYPT 2024, 30th International Conference on the Theory and Application of Cryptology and Information Security, Kolkata, India, December 9-13, 2024, Proceedings, Part VII*, volume 15490 of *Lecture Notes in Computer Science*, pages 301–329. Springer, 2024. doi:10.1007/978-981-96-0941-3_10.

[Zha12] Haibin Zhang. Length-Doubling Ciphers and Tweakable Ciphers. In Feng Bao, Pierangela Samarati, and Jianying Zhou, editors, *Applied Cryptography and Network Security - 10th International Conference, ACNS 2012, Singapore, June 26-29, 2012. Proceedings*, volume 7341 of *Lecture Notes in Computer Science*, pages 100–116. Springer, 2012. doi:10.1007/978-3-642-31284-7_7.

[ZNW21] Itamar Zimerman, Eliya Nachmani, and Lior Wolf. Recovering AES Keys with a Deep Cold Boot Attack. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 12955–12966. PMLR, 2021. Available from: http://proceedings.mlr.press/v139/zimerman21a.html.

[ZS20] Mojtaba Zaheri and Babak Sadeghiyan. SMT-based cube attack on round-reduced Simeck32/64. *IET Inf. Secur.*, 14(5):604–611, 2020. doi:10.1049/IET-IFS.2019.0496.

[ZZ18] Pei Zhang and Wenying Zhang. Differential Cryptanalysis on Block Cipher Skinny with MILP Program. *Secur. Commun. Networks*, 2018:3780407:1–3780407:11, 2018. doi:10.1155/2018/3780407.

A Selected Active Cell Counts for Differential Cryptanalysis

In Tables 9 to 15 we collect results from our MILP runs on several variants of `Vistrutah`, as well as other ciphers. For triples (F, π_0, π_2), F denotes a mixing layer, while π_0 and π_1 are the byte shuffles for even and odd numbered key slices.

Table 9: `Vistrutah.A`, `Pholkos` and `ASURA`, $s = 2$: Minimal number of active S-Boxes for single-key differential characteristics. (Selected examples.)

$r = 2$	4	6	8	10	12	14	16
`Vistrutah.A`-$\Phi/\nu/\zeta/\chi$	5	25	35	60	80	100	110	135
`Vistrutah.A`-ζ^{-1}	5	25	45	61	80	96	115	131
`Vistrutah.A`-ξ	5	15	45	62	75	100	118	135
`Pholkos`	5	25	35	60	80	100	110	135
`ASURA`	5	25	45	61	80	96	115	131

Table 10: `Vistrutah.A`, `Pholkos` and `ASURA`, $s = 2$: Minimal number of active S-Boxes for pure related-key differential characteristics.

$r = 2$	4	6	8	10	12	14	16
`Vistrutah.A`-(Φ, p_4, p_5)	1	6	19	33	43	53	64	74
`Vistrutah.A`-$(\Phi, \sigma_9, \sigma_5)$	1	9	20	33	40	51	63	75
`Vistrutah.A`-(ν, p_4, p_5)	1	8	22	34	43	53	63	72
`Vistrutah.A`-$(\nu, \sigma_9, \sigma_5)$	1	9	22	33	41	51	62	72
`Vistrutah.A`-(ζ, κ, κ)	1	7	23	34	43	53	62	72
`Vistrutah.A`-(ζ, p_4, p_5)	1	7	16	33	43	52	62	72
`Vistrutah.A`-$(\zeta, \sigma_5, \sigma_{10})/(\zeta, \sigma_9, \sigma_5)$	1	7	22/24	33	43	52	61	70
`Vistrutah.A`-(ζ^{-1}, p_4, p_5)	1	10	26	40	55	70	83	100
`Vistrutah.A`-$(\zeta^{-1}, \sigma_5, \sigma_9)$	1	10	22	37	47	56	72	87
`Vistrutah.A`-$(\zeta^{-1}, \sigma_5, \sigma_{10})$	1	8	20	36	40	56	71	81
`Vistrutah.A`-(ξ, p_4, p_5)	1	10	23	38	51	67	84	96
`Vistrutah.A`-$(\xi, \sigma_5, \sigma_{10})$	1	8	20	35	47	59	74	83
`Vistrutah.A`-(χ, p_4, p_5)	1	7	24	33	42	52	62	72
`Vistrutah.A`-$(\chi, \sigma_9, \sigma_5)$	1	7	20	30	37	49	58	67
`Pholkos`	1	6	15	20	25	30	35	40
`ASURA`	1	10	26	38	47	60	71	87

Table 11: `Vistrutah.B` and `Vistrutah.A`-3R, $s = 2$: Minimal number of active S-Boxes for for single-key and pure related-key differential characteristics.

	RK	$r = 3$	6	9	12	15	18
`Vistrutah.A`-3R	×	9	40	59	80	101	122
`Vistrutah.A`-3R-(ζ, κ, κ)	✓	2	18	36	50	63	78
`Vistrutah.A`-3R-(ζ, p_4, p_5)	✓	2	18	36	49	62	75
`Vistrutah.A`-3R-(ζ^{-1}, p_4, p_5)	✓	2	10	33	51	71	90
`Vistrutah.B`-AMDS	×	9	38	51	82	93	124
`Vistrutah.B`-$(\text{AMDS}, \kappa, \kappa)$	✓	3	16	37	55	70	86
`Vistrutah.B`-$(\text{AMDS}, \phi, \phi)$	✓	3	18	38	54	70	84
`Vistrutah.B`-$(\text{AMDS}, \sigma_5, \sigma_9)$	✓	3	21	40	60	80	95
`Vistrutah.B`-MDS	×	9	50	66	100	120	151
`Vistrutah.B`-$(\text{MDS}, \kappa, \kappa)$	✓	3	19	36	56	69	87
`Vistrutah.B`-(MDS, ϕ, ϕ)	✓	3	22	40	56	72	87
`Vistrutah.B`-$(\text{MDS}, \sigma_5, \sigma_9)$	✓	3	20	40	59	78	96

Table 12: Vistrutah and Pholkos, $s = 4$: Minimal number of active S-Boxes for cell-wise differential characteristics.

$r = 2$	4	6	8	10	12	14	16
Vistrutah.A-$\Phi/\nu/\zeta$	5	25	45	80	130	150	170	205
Vistrutah.A-ζ^{-1}	5	25	60	80	100	120	140	160
Vistrutah.A-ξ	5	25	65	106	128	150	185	221
Vistrutah.A-χ	5	25	55	90	120	150	185	215
Pholkos	5	25	45	80	130	150	170	205

Table 13: Vistrutah.A and Pholkos, $s = 4$, with stretched 256-bit keys: Minimal number of active S-Boxes for pure related-key differential characteristics. * The reason for which certain cell counts in the pure related-key model are higher than in the single-key model is explained in Remark 2.

$r = 2$	4	6	8	10	12
Vistrutah.A-copy–(Φ, p_4, p_5)	2	12	38	66	86	106
Vistrutah.A-copy–$(\Phi, \sigma_9, \sigma_5)$	2	18	40	66	80	102
Vistrutah.A-copy–(ν, p_4, p_5)	2	20	45	65	84	102
Vistrutah.A-copy–$(\nu, \sigma_5, \sigma_9)$	2	16	46*	63	80	102
Vistrutah.A-copy–(ζ, p_5, p_4)	2	19	45	65	82	101
Vistrutah.A-copy–$(\zeta, \sigma_9, \sigma_5)$	2	20	40	65	85	102
Vistrutah.A-copy–$(\zeta, \sigma_5, \sigma_{10})$	2	20	45	63	80	97
Vistrutah.A-copy–(ξ, p_4, p_5)	2	17	39	60	77	111
Vistrutah.A-copy–$(\xi, \sigma_5, \sigma_9)$	2	16	36	63	91	107
Vistrutah.A-copy–(χ, p_4, p_5)	2	19	46	80	109	≤ 140
Vistrutah.A-copy–$(\chi, \sigma_9, \sigma_5)$	2	19	47	73	95	≤ 124
Vistrutah.A-Φ–(Φ, p_4, p_5)	2	19	55*	84*	109	138
Vistrutah.A-Φ–(ν, ϕ, ϕ)	2	27*	58*	83*	113	142
Vistrutah.A-Φ–$(\nu, \kappa, \kappa)/(\nu, \kappa, \kappa^{-1})$	2	27*	59*	84*	109/110	137
Vistrutah.A-Φ–$(\nu, \sigma_5, \sigma_9)$	2	27*	54*	83*	109	138
Vistrutah.A-Φ–$(\nu, \sigma_5, \sigma_{10})$	2	27*	55*	79	108	136
Vistrutah.A-Φ–$(\zeta^{-1}, \kappa, \kappa)$	2	20	57	84?	107	135
Vistrutah.A-Φ–$(\zeta^{-1}, \sigma_5, \sigma_9/\sigma_{10})$	2	20	62*	93*	116	≤ 145
Vistrutah.A-Φ–(ζ, κ, κ)	2	20	61*	91*	118	142
Vistrutah.A-Φ–$(\zeta, \kappa, \kappa^{-1})$	2	20	59*	90*	114	139
Vistrutah.A-Φ–$(\zeta, \sigma_5, \sigma_9)$	2	20	59*	83*	107	131
Vistrutah.A-Φ–$(\zeta, \sigma_5, \sigma_{10})$	2	20	60*	84*	113	138
Vistrutah.A-Φ–$(\xi, \phi, \phi)/(\xi, \kappa, \kappa)$	2	20	50	81	109	≤ 136
Vistrutah.A-Φ–(ξ, p_4, p_5)	2	20	52	93	117	≤ 147
Vistrutah.A-Φ–$(\xi, \sigma_5, \sigma_9)$	2	20	57	94	120	145
Vistrutah.A-Φ–$(\chi, \phi, \phi))$	2	20	56*	76	109	133
Vistrutah.A-Φ–(χ, κ, κ)	2	20	59*	85	114	139
Vistrutah.A-Φ–(χ, p_4, p_5)	2	20	59*	89	122	≤ 149
Vistrutah.A-Φ–$(\chi, \sigma_5, \sigma_9)$	2	20	63*	89	117	≤ 145
Vistrutah.A-1AES–(ν, ϕ, ϕ)	2	15	41	66	91	113
Vistrutah.A-1AES–$(\nu, \sigma_5, \sigma_9)$	2	15	40	63	83	96
Vistrutah.A-1AES–$(\nu, \sigma_5, \sigma_{10})$	2	15	43	55	79	92
Vistrutah.A-1AES–$(\zeta, \sigma_5, \sigma_9)$	2	18	48*	66	93	≤ 114
Vistrutah.A-1AES–$(\zeta, \sigma_5, \sigma_{10})$	2	18	45	66	87	105
Vistrutah.A-1AES–(ζ, p_5, p_4)	2	18	38	66	91	110
Vistrutah.A-2AES–$(\nu, \phi, \phi)/(\nu, \kappa, \kappa)$	5	16	46*	65/64	84	≤ 107
Vistrutah.A-2AES–$(\zeta, \sigma_5, \sigma_{10})$	5	16	43	64	79	≤ 108
Vistrutah.A-2AES–$(\zeta, \sigma_5, \sigma_9)$	5	25	46*	66	85	≤ 103
Pholkos	4	29	56	83	96	115

Table 14: `Vistrutah.A` and `Pholkos`, $s = 4$, with 512-bit keys: Minimal number of active S-Boxes for related-key differential characteristics. (Selected examples.)

	$r = 2$	4	6	8	10	12	14
`Vistrutah.A`–(Φ, p_4, p_5)	1	10	27	49	66	89	—
`Vistrutah.A`–$(\Phi, \sigma_9, \sigma_5)$	1	10	29	49	66	87	—
`Vistrutah.A`–(ν, ϕ, ϕ)	1	10	27	42	59	82	103
`Vistrutah.A`–$(\nu, \sigma_5, \sigma_9)$	1	9	28	48	61	83	106
`Vistrutah.A`–$(\zeta^{-1}, \sigma_5, \sigma_9)$	1	10	37	53	76	92	—
`Vistrutah.A`–$(\zeta^{-1}, \sigma_5, \sigma_{10})$	1	10	37	53	72	87	—
`Vistrutah.A`–(ζ, p_5, p_4)	1	7	16	42	64	84	—
`Vistrutah.A`–(ζ, κ, κ)	1	6	19	46	59	80	—
`Vistrutah.A`–$(\zeta, \sigma_5, \sigma_9)$	1	10	31	51	68	88	108
`Vistrutah.A`–$(\zeta, \sigma_5, \sigma_{10})$	1	10	31	47	68	86	110
`Vistrutah.A`–$(\xi, \phi, \phi))$	1	10	20	27	34	41	50
`Vistrutah.A`–(ξ, κ, κ)	1	10	18	27	34	40	50
`Vistrutah.A`–(ξ, p_4, p_5)	1	10	31	53	68	88	—
`Vistrutah.A`–$(\xi, \sigma_5, \sigma_9)$	1	10	35	57	76	89	—
`Vistrutah.A`–$(\chi, \phi, \phi))$	1	8	25	44	63	83	106
`Vistrutah.A`–(χ, κ, κ)	1	10	26	50	66	87	106
`Vistrutah.A`–(χ, p_4, p_5)	1	7	25	48	74	92	117
`Vistrutah.A`–$(\chi, \sigma_5, \sigma_9)$	1	7	24	49	67	84	108
`Pholkos`	1	8	26	40	50	60	80

Table 15: `Vistrutah.B` and `Vistrutah.A`-3R, $s = 4$, with stretched 256-bit keys and full 512-bit keys: Minimal number of active S-Boxes for single-key and pure related-key differential characteristics.

	RK	Key	$r = 3$	6	9	12	15	18
`Vistrutah.A-3R`	×	All	9	45	66	90	114	138
`Vistrutah.A-Φ-3R-`(ν, κ, κ)	✓	256	9	48	89	126	163	—
`Vistrutah.A-Φ-3R-`(ν, ϕ, ϕ)	✓	256	9	48	83	126	166	—
`Vistrutah.A-3R-`(ν, κ, κ)	✓	512	3	22	44	66	84	100
`Vistrutah.A-3R-`(ν, ϕ, ϕ)	✓	512	3	22	45	69	90	105
`Vistrutah.B-AMDS`	×	All	9	46	65	112	140	180
`Vistrutah.B-circ-`$(\text{AMDS}, \kappa, \kappa)$	✓	256	8	41	65	90	123	—
`Vistrutah.B-circ-`$(\text{AMDS}, \sigma_9, \sigma_9)$	✓	256	8	41	77	97	131	—
`Vistrutah.B-circ-`$(\text{AMDS}, \sigma_9, \sigma_5)$	✓	256	6	32	65	94	110	—
`Vistrutah.B-Φ-`$(\text{AMDS}, \kappa, \kappa)$	✓	256	10	56	89	129	—	—
`Vistrutah.B-Φ-`$(\text{AMDS}, \sigma_9, \sigma_9)$	✓	256	10	56	92	126	162	—
`Vistrutah.B-Φ-`$(\text{AMDS}, \sigma_9, \sigma_5)$	✓	256	10	52	97	≥ 120	≥ 139	—
`Vistrutah.B-`$(\text{AMDS}, \kappa, \kappa)$	✓	512	3	16	37	55	70	86
`Vistrutah.B-`$(\text{AMDS}, \sigma_9, \sigma_9)$	✓	512	3	21	32	56	70	84
`Vistrutah.B-`$(\text{AMDS}, \sigma_9, \sigma_5)$	✓	512	3	21	40	60	80	95
`Vistrutah.B-MDS`	×	All	9	55	74	120	164	208
`Vistrutah.B-Φ-`$(\text{MDS}, \kappa, \kappa)$	✓	256	10	55	92	125	161	—
`Vistrutah.B-Φ-`$(\text{MDS}, \sigma_9, \sigma_9)$	✓	256	10	57	92	124	156	—
`Vistrutah.B-`$(\text{MDS}, \kappa, \kappa)$	✓	512	3	31	56	91	119	137
`Vistrutah.B-`$(\text{MDS}, \sigma_9, \sigma_9)$	✓	512	3	30	61	96	125	—

B Detailed Performance Data

Tables 16 and 17 collect timings for our implementations. The first timing for 512-bit Vistrutah.B-AMDS and Vistrutah.B-MDS encryption on NEON and decryption on AVX is of the code that exploits the fact that the key shuffle consists of four equal in-line permutations as described in Remark 3 and the value in parentheses is for the generic code that works with any key shuffle.

Table 16: Timings of selected versions of the ciphers considered in this paper, on an Apple Silicon M3 computer running at 3.67Ghz. The C compiler is Clang version 17.0.

256-Bit Cipher	r	Time (ns)	
		Enc.	Dec.
Vistrutah.A-2R-ζ^{-1}	8	2.93	3.11
Vistrutah.A-2R-ζ^{-1}	10	3.42	4.09
Vistrutah.A-2R-ζ^{-1}	12	3.91	5.07
Vistrutah.A-2R-ζ^{-1}	14	4.39	6.29
Vistrutah.A-2R-ζ^{-1}	16	4.91	7.32
Vistrutah.A-2R-ξ	8	2.93	3.17
Vistrutah.A-2R-ξ	10	3.42	4.15
Vistrutah.A-2R-ξ	12	3.91	5.31
Vistrutah.A-2R-ξ	14	4.39	6.53
Vistrutah.A-2R-ξ	16	5.13	7.63
Vistrutah.A-3R	9	2.44	2.69
Vistrutah.A-3R	12	2.93	3.66
Vistrutah.A-3R	15	3.66	4.94
Vistrutah.A-3R	18	4.70	6.10
Vistrutah.A-3R	21	5.92	7.57
Vistrutah.B-AMDS	9	2.69	2.69
Vistrutah.B-AMDS	12	3.91	3.72
Vistrutah.B-AMDS	15	4.76	4.94
Vistrutah.B-AMDS	18	6.16	6.29
Vistrutah.B-AMDS	21	7.87	7.81
Vistrutah.B-MDS	9	2.69	2.81
Vistrutah.B-MDS	12	4.09	4.09
Vistrutah.B-MDS	15	5.19	5.37
Vistrutah.B-MDS	18	6.77	6.96
Vistrutah.B-MDS	21	8.48	8.48
Pholkos	8	3.48	4.52
Pholkos	10	4.46	5.86
Pholkos	12	5.49	7.08
Pholkos	14	6.59	8.48
Pholkos	16	7.81	9.77
Ghidle	9	2.71	2.89
Ghidle	12	3.91	4.09
Ghidle	15	5.07	5.31
Ghidle	18	6.47	6.77
Ghidle	21	7.81	8.12
ASURA	8	3.17	4.94
ASURA	10	4.03	6.65
ASURA	12	5.00	8.18
ASURA	14	6.04	10.25
ASURA	16	7.20	11.90

512-Bit Cipher	r	Time (ns)	
		Encryption	Dec.
Vistrutah.A-2R-ζ	10	6.29	8.36
Vistrutah.A-2R-ζ	12	7.69	10.25
Vistrutah.A-2R-ζ	14	9.03	12.39
Vistrutah.A-2R-ζ	16	10.31	14.65
Vistrutah.A-2R-ζ	18	11.78	16.66
Vistrutah.A-2R-ζ^{-1}	10	6.31	8.36
Vistrutah.A-2R-ζ^{-1}	12	7.69	10.31
Vistrutah.A-2R-ζ^{-1}	14	9.01	12.41
Vistrutah.A-2R-ζ^{-1}	16	10.33	14.71
Vistrutah.A-2R-ζ^{-1}	18	11.78	16.66
Vistrutah.A-2R-ν	10	6.47	8.48
Vistrutah.A-2R-ν	12	7.81	10.38
Vistrutah.A-2R-ν	14	9.28	12.57
Vistrutah.A-2R-ν	16	10.93	14.71
Vistrutah.A-2R-ν	18	12.70	16.85
Vistrutah.A-2R-ξ	10	7.39	9.70
Vistrutah.A-2R-ξ	12	8.91	12.21
Vistrutah.A-2R-ξ	14	10.68	14.65
Vistrutah.A-2R-ξ	16	12.63	17.09
Vistrutah.A-2R-ξ	18	14.47	19.65
Vistrutah.A-3R-ν	18	9.70	12.76
Vistrutah.A-3R-ν	21	11.78	15.32
Vistrutah.A-3R-ν	24	14.16	17.70
Vistrutah.A-3R-ν	27	16.42	20.45
Vistrutah.A-3R-ν	30	18.68	23.13
Vistrutah.B-AMDS	15	7.51 (8.00)	7.32
Vistrutah.B-AMDS	18	9.09 (9.77)	8.73
Vistrutah.B-AMDS	21	10.80 (11.78)	10.31
Vistrutah.B-AMDS	24	12.63 (13.85)	11.96
Vistrutah.B-AMDS	27	14.40 (15.87)	13.51
Vistrutah.B-MDS	12	9.03 (11.54)	9.34
Vistrutah.B-MDS	15	12.45 (14.53)	12.45
Vistrutah.B-MDS	18	15.81 (17.58)	15.69
Vistrutah.B-MDS	21	19.29 (20.94)	19.04
Vistrutah.B-MDS	24	23.07 (25.57)	22.64
Pholkos	10	16.05	19.35
Pholkos	12	19.65	23.19
Pholkos	14	23.32	27.16
Pholkos	16	26.92	31.13
Pholkos	18	30.52	35.10
Pholkos	20	34.18	39.12
Ghidle	15	8.91	8.91
Ghidle	18	10.68	11.11
Ghidle	21	12.57	12.51
Ghidle	24	14.28	14.77
Ghidle	27	16.17	16.11

Table 17: Timings of the selected version of `Vistrutah`, alternative versions, and other ciphers on an Intel Core i7 13700K CPU running at 5 GHz. The compiler is GCC 11.0.4. The presence of two timings for `Vistrutah.B`-512 decryption is explained in Remark 3.

256-Bit Cipher	r	Time (ns)	
		Enc.	Dec.
`Vistrutah.A-2R-`$\boldsymbol{\zeta^{-1}}$	8	3.51	4.25
`Vistrutah.A-2R-`$\boldsymbol{\zeta^{-1}}$	10	4.63	5.58
`Vistrutah.A-2R-`$\boldsymbol{\zeta^{-1}}$	12	6.14	7.60
`Vistrutah.A-2R-`$\boldsymbol{\zeta^{-1}}$	14	7.35	8.65
`Vistrutah.A-2R-`$\boldsymbol{\zeta^{-1}}$	16	8.84	10.60
`Vistrutah.A-2R-`ξ	8	3.62	5.34
`Vistrutah.A-2R-`ξ	10	4.59	6.98
`Vistrutah.A-2R-`ξ	12	5.93	9.18
`Vistrutah.A-2R-`ξ	14	7.38	12.27
`Vistrutah.A-2R-`ξ	16	8.62	13.69
`Vistrutah.A-3R`	9	2.94	3.80
`Vistrutah.A-3R`	12	4.34	5.88
`Vistrutah.A-3R`	15	5.78	7.41
`Vistrutah.A-3R`	18	7.58	9.57
`Vistrutah.A-3R`	21	9.31	13.37
`Vistrutah.B-AMDS`	9	2.85	3.90
`Vistrutah.B-AMDS`	12	4.29	5.33
`Vistrutah.B-AMDS`	15	5.63	6.85
`Vistrutah.B-AMDS`	18	6.98	8.76
`Vistrutah.B-AMDS`	21	8.62	10.77
`Vistrutah.B-MDS`	9	3.86	4.94
`Vistrutah.B-MDS`	12	5.92	7.42
`Vistrutah.B-MDS`	15	8.25	10.23
`Vistrutah.B-MDS`	18	10.89	13.53
`Vistrutah.B-MDS`	21	13.94	16.68
`Pholkos`	8	4.72	7.51
`Pholkos`	10	6.49	10.67
`Pholkos`	12	7.81	11.88
`Pholkos`	14	9.40	14.38
`Pholkos`	16	10.63	15.06
`Ghidle`	9	2.97	4.41
`Ghidle`	12	4.39	5.82
`Ghidle`	15	6.18	7.25
`Ghidle`	18	7.92	9.08
`Ghidle`	21	8.68	10.38
`ASURA`	8	7.65	8.86
`ASURA`	10	9.85	11.33
`ASURA`	12	12.01	13.63
`ASURA`	14	14.77	16.95
`ASURA`	16	17.80	19.50

512-Bit Cipher	r	Time (ns)	
		Enc.	Decryption
`Vistrutah.A-2R-`$\boldsymbol{\zeta}$	10	9.01	13.81
`Vistrutah.A-2R-`$\boldsymbol{\zeta}$	12	11.82	17.08
`Vistrutah.A-2R-`$\boldsymbol{\zeta}$	14	13.58	21.82
`Vistrutah.A-2R-`$\boldsymbol{\zeta}$	16	16.32	24.97
`Vistrutah.A-2R-`$\boldsymbol{\zeta}$	18	19.38	28.90
`Vistrutah.A-2R-`ζ^{-1}	10	10.65	11.89
`Vistrutah.A-2R-`ζ^{-1}	12	13.60	15.12
`Vistrutah.A-2R-`ζ^{-1}	14	16.70	19.04
`Vistrutah.A-2R-`ζ^{-1}	16	19.83	22.39
`Vistrutah.A-2R-`ζ^{-1}	18	22.79	26.66
`Vistrutah.A-2R-`ν	10	8.80	11.72
`Vistrutah.A-2R-`ν	12	11.13	14.77
`Vistrutah.A-2R-`ν	14	13.37	18.54
`Vistrutah.A-2R-`ν	16	16.08	22.40
`Vistrutah.A-2R-`ν	18	18.95	25.96
`Vistrutah.A-2R-`ξ	10	12.26	13.91
`Vistrutah.A-2R-`ξ	12	15.83	17.65
`Vistrutah.A-2R-`ξ	14	19.29	21.86
`Vistrutah.A-2R-`ξ	16	23.06	25.85
`Vistrutah.A-2R-`ξ	18	26.94	30.07
`Vistrutah.A-3R-`ν	18	13.99	18.87
`Vistrutah.A-3R-`ν	21	17.61	23.37
`Vistrutah.A-3R-`ν	24	20.75	27.76
`Vistrutah.A-3R-`ν	27	24.38	32.15
`Vistrutah.A-3R-`ν	30	27.61	36.56
`Vistrutah.B-AMDS`	15	8.73	9.98 (11.23)
`Vistrutah.B-AMDS`	18	10.80	12.15 (14.52)
`Vistrutah.B-AMDS`	21	13.40	14.50 (16.51)
`Vistrutah.B-AMDS`	24	15.46	16.86 (20.45)
`Vistrutah.B-AMDS`	27	17.43	19.13 (22.49)
`Vistrutah.B-MDS`	12	15.32	14.53 (21.95)
`Vistrutah.B-MDS`	15	19.98	18.51 (26.11)
`Vistrutah.B-MDS`	18	25.86	23.11 (32.19)
`Vistrutah.B-MDS`	21	32.79	27.66 (39.09)
`Vistrutah.B-MDS`	24	39.07	31.93 (45.32)
`Pholkos`	10	13.92	23.32
`Pholkos`	12	17.35	28.30
`Pholkos`	14	21.07	33.83
`Pholkos`	16	23.96	38.71
`Pholkos`	18	27.23	41.96
`Pholkos`	20	29.82	47.40
`Ghidle`	15	11.85	12.24
`Ghidle`	18	14.61	15.57
`Ghidle`	21	17.03	17.13
`Ghidle`	24	19.72	21.03
`Ghidle`	27	21.48	21.30

C Alternative Round Constants for Vistrutah

We use the following primitive polynomial over GF(2) to define a LFSR:

$$X^{128} + X^{97} + X^{62} + X^{39} + 1 \ . \tag{27}$$

We have chosen this polynomial so that we can refresh all bits by clocking the LFSR a limited number of timesin, this case 39 or more. To reduce the circuit depth, we shift and reduce the polynomial by more than one bit at a time. Since the distances between exponents of consecutive monomials (31, 35, 23, and 39 in our case) cannot all be equal, the smallest difference limits the number of bits that can be shifted at once with a circuit of depth 1 XOR. We require the smallest difference to be at least half the total number of bits we shift, ensuring the resulting circuit has a total depth of 2 XORs.

Note that (27) is not the only primitive pentanomial satisfying these requirements. No such trinomials exist. The pentanomial $X^{128}+X^{96}+X^{64}+X^{32}+1 = (X^4+X^3+X^2+1)^{32}$ would allow a reduction circuit of depth 1 XOR, but it is trivially reducible. There is no need to use heptanomials or denser polynomials to refresh all bits.

We tick the LFSR 43 times, in two steps of 22 and 21 ticks. Since 43 and $2^{128}-1$ are coprime, the resulting round constant update function Ψ has a single cycle of length $2^{128}-1$ on the non-zero 128-bit strings. The code in Figure 12 implements Ψ and its inverse. In fact, this is valid C code if the `uint128_t` type is defined in the `stdint.h` header file. It is the same code used to test the functions and precompute the static round constants for software implementations.

```
uint128_t PSI(uint128_t in)
{
    uint128_t s, tmp = in;

    s = tmp >> 106;
    tmp = (tmp << 22) ^ (s << 97) ^ (s << 62) ^ (s << 39) ^ s;

    s = tmp >> 107;
    tmp = (tmp << 21) ^ (s << 97) ^ (s << 62) ^ (s << 39) ^ s;

    return tmp;
}

uint128_t PSIInverse(uint128_t in)
{
    uint128_t s, tmp = in;

    s = tmp & ((1 << 22) - 1);
    tmp = (tmp >> 22) ^ (s << 75) ^ (s << 40) ^ (s << 17) ^ (s << 106);

    s = tmp & ((1 << 21) - 1);
    tmp = (tmp >> 21) ^ (s << 76) ^ (s << 41) ^ (s << 18) ^ (s << 107);

    return tmp;
}
```

Figure 12: Code for the round constant generating function PSI (Ψ) and its inverse.

Note that the variable `s` (after *spill*) is just a 22 or 21 bit string. Both Ψ and its inverse are thus implemented with $3 \cdot 22 + 3 \cdot 21 = 129$ XORs, since the last operation in each folding ("`^ s`", "`^ (s << 106)`", and "`^ (s >> 107)`") simply assigns bit values and does not XOR them because of the (unsigned) shift of `tmp`. Therefore the area of both Ψ and of its inverse is a little less than one S-Box, and does not impact the HW cost evaluations performed in Subsection 5.2.

We finally define the round constants as follows: $\mathfrak{c}_0 = 0$, $\mathfrak{c}_1$ consists of the first 128 bits of the binary expansion of π, $\mathfrak{c}_i = \Psi(\mathfrak{c}_{i-1})$ for i from 2 to $r/2$ included, and $\mathfrak{c}_{r/2+1} = 0$.

D Breakdown of the Costs of the Steps for all the Ciphers

Here we detail the breakdown of the costs for the individual components of the steps for each cipher considered in this paper. These costs fall into two categories: *baseline costs* (excluding round key and round constant computation) and the *round key and round constant computation*. Note that we distinguish between the costs associated to the round constants and the round keys. We now proceed to detail these costs.

1. **Cost of a `Vistrutah.A` step, with $r_{st} = 2$:**
   - *Baseline encryption step:*
     - Two groups of s `AES` round operations.
     - For the cost of the mixing layer, cf. Items **7.** and **8.** below.
   - *Round key and round constant costs:*
     - One group of loads operations to fetch the round constants.
     - The round key update (cf. Item **9.** below) is a group of s in-slice shuffles. Note that these operations can be performed before the following mixing layer, so that:
     - The single addition to add round constant and round key can be usually grouped with other operations, so this is not counted.
   - The decryption step has two extra groups of operations (in the decryption one can fuse only one `AESD`/`AESIMC` pair, so there are four microarchitectural instructions in the data path beside the mixing layer, instead of two). However, on decryption the single addition to add the round constant can be grouped with other additions, at least if $s = 2$.

2. **Cost of a `Vistrutah.A` step, with $r_{st} = 3$:** Only the baseline cost is different, as it requires one extra group of `AES` round operations using a fixed key.

3. **Cost of a `Vistrutah.B` encryption step ($r_{st} = 3$):**
   - *Baseline encryption step cost:*
     - Three groups of s parallel `AES` round operations.
     - Three (for $s = 2$) or two (for $s = 4$) groups to implement the AMDS mixing layer (cf. Items **7.** and **8.** below).
   - A decryption step requires an additional group of key additions.
   - *Round key and round constant costs:* same as for `Vistrutah.A` with $r_{st} = 2$.

4. **Cost of a (untweaked) `Pholkos` encryption step:** Note that in order to perform a fair comparison, we use `Pholkos` only with 256-bit keys and removing the tweak.
   - *Baseline encryption step:*
     - On architectures where the addition of the round key is the last operation in the `AES` instruction, the cost is two groups of `AES` round operations, otherwise an additional group of key additions is needed.
     - One (for $s = 2$) or about three (for $s = 4$) groups to implement the mixing layer (cf. Items **7.** and **8.** below).
   - A `Pholkos` decryption step contains two additional groups of IMC instructions. It is easy to see that they are independent of the IMC group to update the round key, so they can be issued independently and ignored in the cost estimate.
   - *Round key and round constant costs (to be counted twice):*
     - The round key update combines the mixing layer permutation with in-lane shuffles on all lanes. For $s = 2$, this can also be done with a single group of two-register table-lookups, and for $s = 4$, with three groups of operations

on AVX and four four-register table-lookups on NEON with a cost also roughly equivalent to three "normal" groups of operations.
  - Round constants and round keys are added in two groups of operations.

5. **Cost of a Ghidle encryption step:**
   - *Baseline encryption step cost: Five groups including the mixing layer.*
     - Three groups of s AES round operations.
     - Two group to implement the mixing layer (cf. Items 7. and 8. below).
   - A Ghidle decryption step has the same cost as the encryption step.
   - *Round key and round constant costs: Three groups.*
     - The round key update is the same as the mixing layer.
     - Round constants and round keys are added in a group of operations.

6. **Cost of an (untweaked) ASURA step:**
   - *Baseline encryption step cost: Three groups including the mixing layer.*
     - One group: Two parallel AES round operations using a round key.
     - One group: One AES round encryption operation and one inverse, which cannot be issued by a single instruction, but can be issued in parallel.
     - One group to implement the mixing layer (cf. Item 7. below).
   - An ASURA decryption step requires two additional round key additions instructions with respect to the encryption step.
   - *Round key and round constant costs (to be counted twice):*
     - The key schedule update can be done with a single group of operations, as in the two-slice versions of Vistrutah and Pholkos.
     - Round constants and round keys are added in a group of operations.

7. **Costs of mixing layer updates, 256-bit versions:** All shuffle-based mixing layers can be implemented with a single group of operations on NEON, either with vzip/vuzp/vtrn instructions or with two-input table lookups.

   The AMDS matrix used in Ghidle and Vistrutah.B requires three groups of operations. Let a, b, c, and d be 64-bit elements, stored in two 128-bit register are $a\|b$ and $c\|d$ (here a and c are the least significant halves). In the first operation group we compute $(a \oplus c)\|(b \oplus d)$ and rotate the two input vectors to obtain $b\|a$ and $d\|c$. In the second group, by two more XORs we obtain $(a \oplus b \oplus c)\|(a \oplus b \oplus d)$ and $(a \oplus c \oplus d)\|(b \oplus c \oplus d)$. The correct ordering of the output requires a third group of operations, consisting of two rotations.

   Applying the MDS matrix to the four half states requires three groups of operations on NEON: the ζ shuffle (20), an inverse MixColumns operation, and the inverse shuffle ζ^{-1}, i.e., (2). The inverse operation has the same cost.

   On AVX the cost of the MDS matrix is higher. ζ is a single group of operations, but ζ^{-1} needs two groups of operations, hence an encryption step requires one more group. For decryption, we need to apply MixColumns, which is emulated by applying AESDECLAST and then AESENC with zero keys, in order. Therefore decryption requires two more groups of operations.

   On AVX, ν and ζ^{-1} require two groups of operations. ξ, χ and their inverses can be implemented with a group of blend and a group of shuffle instruction.

8. **Costs of mixing layer updates, 512-bit versions:**
   - Pholkos' Φ needs three groups of operation on AVX. On NEON it can be implemented by considering the state as a 4×4-matrix of words (the rows),

which is then transposed, subjected to a `ShiftRows`-like operation, and then transposed again. This results in 5 groups, which would make the cost of the key schedule 6 groups. It turns out that on NEON the mixing layer and the key scheduling function can be implemented using four four-register table-lookups, for a cost roughly corresponding to 3 groups. This alone makes both encryption and decryption more than 20 % faster than with the previous approach.
- ν, ζ and ζ^{-1}, as well as the AMDS matrix require two groups on NEON. One extra group is needed on AVX.
- The MDS matrix requires the application of ζ, the inverse `MixColumns` for encryption and `MixColumns` for decryption, and finally ζ^{-1}, in order. On NEON this consists of $2 + 1 + 2 = 5$ groups for both encryption and decryption. On AVX this consists of $2 + 1 + 3 = 6$ groups for encryption and $2 + 2 + 3 = 7$ groups for decryption.
- ξ and χ require three groups.

9. Costs of key schedule updates:

- p_4, p_5 and other arbitrary permutations: one group. On some architectures one or two registers must be reserved to store the description of the permutation.
- Slice rotations: one group, no additional registers to be allocated.
- The `Pholkos` key scheduling has already been discussed.

E Detailed Cryptanalysis of `Vistrutah`

We give here the full details of the cryptanalysis of `Vistrutah`. We measure the memory complexity of the attacks in *states*, i.e., blocks of memory that have the same size of a state of the cipher, namely 256 bits for `Vistrutah`-256 and 512 bits for `Vistrutah`-512. The unit for the time complexity of the attacks is the EE, i.e., the time of an encryption or decryption operation. In some cases, the time complexity also contains MAs.

E.1 Integral Cryptanalysis

The integral or square attack has been introduced by Lars Knudsen and the future designers of `Rijndael` in [DKR97]. Since the `AES` has been designed to resist such attacks, it is only natural to start our cryptanalysis with this attack.

E.1.1 Integral Attack on 7-round `Vistrutah`-256

E.1.1.1 Distinguisher

A five-round integral trail covering Rounds 1–5 is shown in Figure 13. The trail can be seen to propagate from any byte in W_0^2 to all bytes of P_0 and W_0^4 and from there to the first two inverse diagonals of Z_0^5 and Z_1^5, ending in the balanced states W_0^5 and W_1^5.

A key-recovery trail that appends two rounds to the distinguisher is depicted in Figure 14. We define $\widetilde{K}_0^6$ to be the equivalent subkey that, after being transformed through `MixColumns`, is added to W_0^6 to obtain the first two columns of X_0^7 and X_1^7 after applying the mixing layer. Each byte of X_0^6 or X_1^6 should sum to zero, so recovering any byte will suffice. Figure 14 and the following description focuses on recovering $X_0^6[0]$.

We start by guessing five key bytes: $\widetilde{K}_0^6[0]$, $\widehat{K}_0^7[0, 13]$, and $\widehat{K}_1^7[0, 13]$.

E.1.1.2 Attack

We begin with a naïve attack that is accelerated using the FFT technique later. It works as follows:

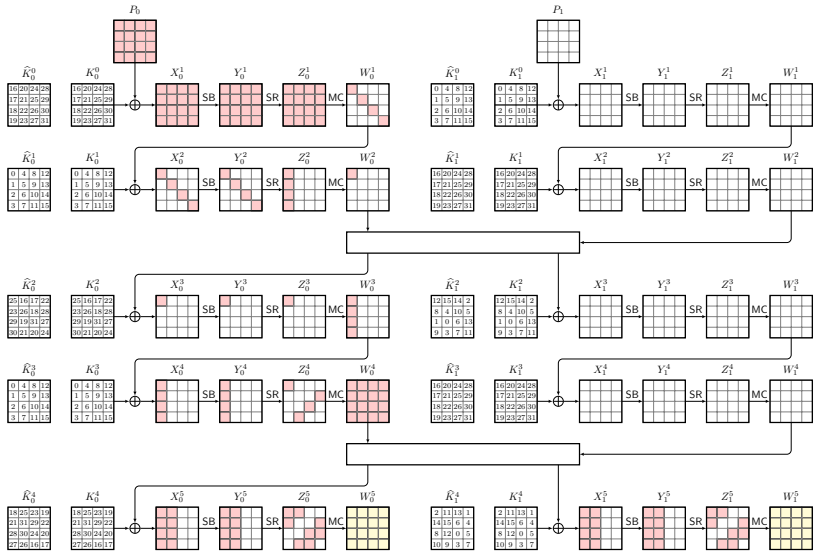


Figure 13: Integral distinguisher on five-round Vistrutah-256.

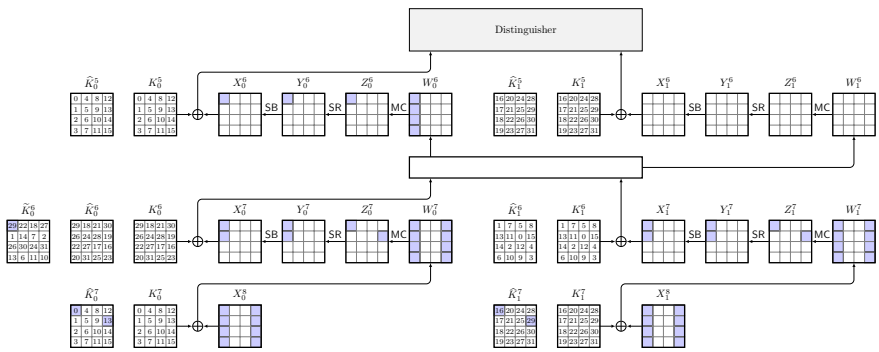


Figure 14: The key-recovery rounds of the seven-round integral attack on Vistrutah-256.

1. Initialize empty lists of eight-bit sums, one for each key candidate $\mathcal{K}$.
2. Form s structures of 2^{128} plaintexts $\mathcal{P}$ each that iterate over all 16 bytes of P_0.
3. Ask for their corresponding encryptions after seven rounds and invert the final `MixColumns` operation to obtain $\widehat{W}$ from each ciphertext C.
4. For each state $\widehat{W}$ and all key candidates:
   (a) Partially decrypt to $X_0^6[0]$ and add the result to a sum for the current key candidate.
   (b) Discard all key candidates with nonzero sums.
   (c) Repeat the process for all structures $\ell \in \{1, \ldots, s\}$ and keep only key candidates that produce a zero sum for all s structures.
5. Repeat the process for different bytes of X_0^6 to recover a different set of the key space, e.g. in X_0^6 to recover other parts of the key.
6. After this stage, return the key candidates that passed all tests.

E.1.1.3 Complexity

The probability that any of the $2^{5\cdot 8}$ key guesses survives the key-space tests is $2^{-8\cdot s}$. For $s = 6$, we expect a probability of approximately 2^{-8} for any wrong key to pass, and 2^{-6} that any wrong key pass four iterations for the four rows of the final subkey. Therefore, we expect only the correct key and very few false positives, if any, to remain at the end. The attack has following costs:

- $s \cdot 2^{128}$ encryptions and $2 \cdot s \cdot 2^{128}$ MAs to the huge tables $\mathcal{C}_i$ for storing and reading the ciphertexts,
- $2 \cdot s \cdot 2^{128} \cdot 2^k$ read accesses into $\mathcal{C}_i$,
- $2 \cdot s \cdot 2^{128} \cdot 2^k$ partial encryptions of 5 out of 256 S-boxes and additions, that are at most $2s \cdot 2^{5\cdot 8+128} \cdot 5/256 \approx s \cdot 2^{293.4}$ EEs, and
- a negligible amount of surviving keys to store in $\mathcal{K}$.

Thus, the data-collection phase needs $6 \cdot 2^{128} \approx 2^{130.6}$ EEs and $2^{131.6}$ MAs. The attack needs $2 \cdot 6 \cdot 2^{128+5\cdot 8} \cdot 5/256 \approx 2^{165.9}$ EEs and $2 \cdot 6 \cdot 2^{128+5\cdot 8} \approx 2^{171.6}$ MAs. The memory is dominated by storing 2^{128} texts at a time. Summarizing, the complexities are:

- Data-collection phase: $2^{130.6}$ EEs, $2^{131.6}$ MAs, $2^{130.6}$ CPs.
- Memory: 2^{128} states.
- Attack phase: $2^{165.9}$ EEs and $2^{171.6}$ MAs to large tables.

Assuming that each MA is as expensive as a full 10-round encryption, we obtain an attack complexity of at most $2^{171.6}$ EEs.

E.1.1.4 Using the FFT

In the following, we describe how the attack on 7-round `Vistrutah` can be improved by using the FFT technique. Collard et al. [CSQ07] applied the FFT technique to linear cryptanalysis, and Todo and Aoki [TA14] showed how to use it to speed up integral cryptanalysis as well. The FFT technique reduces the complexity of computing several sums over all 2^t texts and all involved 2^k subkeys, by separating the involved key material into $k = k_1 + k_2$ bits, where k_1 denotes the bits directly before the final S-box and k_2 the remaining involved bits. We have $2^t = 2^{128}$ plaintext values, $k_1 = 8$ bits for the byte in $\widehat{K}^9$ and $k_2 = 32$ for the remaining 20 bytes of guessed keys. The four steps described by Todo and Aoki are:

- A circuit-evaluation step that creates a $2^{k_1} \cdot 2^{k_2}$-dimensional vector with $c_1 = 2^{k_1} \cdot k_1 \cdot k_2 \cdot 2^{k_2}$ additions. This yields $c_1 = 2^8 \cdot 8 \cdot 32 \cdot 2^{32} \approx 2^{48}$ additions in our case.
- A ciphertext-evaluation step of $c_2 = t \cdot 2^t$ additions — in our case $c_2 = 2^{135}$.
- A key-recovery step of $2^{k_1} \cdot k_1 \cdot 2^{k_2}$ multiplications, or $2^{k_1} \cdot k_1 \cdot 2^{k_2} \cdot 2^5$ additions, plus $2^{k_1} \cdot k_1 \cdot k_2 \cdot 2^{k_2}$ additions, i.e., $c_3 = (2^8 \cdot 8 \cdot 2^{32}) \cdot (2^5 + 32) \approx 2^{49}$ additions.

The final effort must then be repeated s times to filter the keys, and four times in our case for the four rows of the final subkey to obtain candidates for the full key, which yields $32 \cdot (2^{48} + 6 \cdot 2^{49}) \approx 2^{56.7}$ additions. Note that those approaches usually neglect MAs that can be expensive in our case. Therefore, we expect approximately the same number of read accesses to $\mathcal{C}$. Additions are less expensive than EEs, but they require MAs — we approximate the latter with EEs.

That said, the attack's time complexity becomes dominated by collecting and storing the data, which we approximate by $2^{135} + 2^{130.6} + 2^{131.6} + 2^{56.7} \approx 2^{135.2}$ EEs. Moreover, we can reduce the memory significantly by only storing:

- The sums for the key candidates, i.e., 128-bit values each for 32 bytes of X^6, s structures and 2^{40} candidates at a time, i.e. $6 \cdot 32 \cdot 2^{40} \cdot 128/256 \approx 2^{46.6}$ states.

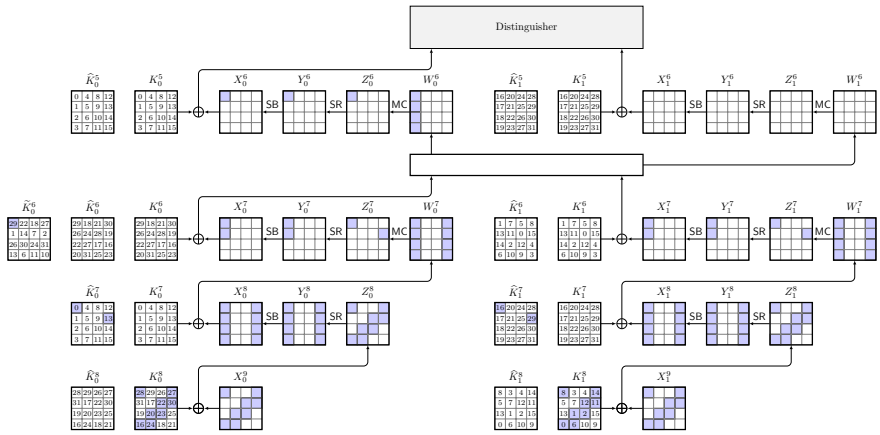


Figure 15: The key-recovery rounds of the eight-round integral attack on Vistrutah-256.

- 128-bit counters for the numbers of occurrences of each 32-bit value at $\widehat{W}^7$ that corresponds to a column at W^6, i.e., $s \cdot 8 \cdot 2^{32} \cdot 128/256 \approx 2^{36.6}$ states at a time.

Therefore, we reduce the memory complexity to $2^{46.6} + 2^{36.6} \approx 2^{46.6}$ states. Summarizing:

- Data-collection phase: $2^{130.6}$ EEs, $2^{135} + 2^{131.6} = 2^{135.1}$ MAs, $2^{130.6}$ CPs.
- Memory: $2^{46.6}$ states.
- Attack phase: $2^{132.2}$ EEs.

E.1.2 Integral Attack on 8-round Vistrutah-256

In the following, we reuse the five-round integral trail of Figure 13, covering Rounds 1–5, to mount an integral attack on eight rounds.

E.1.2.1 Attack

The key-recovery trail depicted in Figure 15 appends three rounds to the distinguisher. We define $\widetilde{K}_0^6$ as the equivalent subkey that, when transformed through MixColumns is added to W_0^6 to obtain the first two columns of X_0^7 and X_1^7 after applying the mixing layer. Each byte of X_0^6 or X_1^6 should sum to zero, so recovering any byte will suffice. We now focus on recovering $X_0^6[0]$. We guess 21 key bytes:

- $\widetilde{K}_0^6[0]$,
- $\widehat{K}_i^7[0, 13]$ for $i = 0, 1$, and
- $K_0^8[0, 3, 6, 7, 9, 10, 12, 13]$ for $i = 0, 1$.

In the following, we describe a naive attack that may be sped up using the FFT and partial-sum techniques. We conduct the following steps:

1. Initialize an empty list of key candidates $\mathcal{K}$.
2. Form s structures of 2^{128} (sub)structures $\mathcal{P}$ each that iterate over all 16 bytes of P_0.
3. Ask for their corresponding encryptions after eight rounds, and store the results into a list $\mathcal{C}_\ell$ for $\ell \in \{1, \ldots, s\}$.
4. For all key candidates and all anti-diagonals $\mathcal{ID}_{0,3}(C_0) \| \mathcal{ID}_{0,3}(C_1)$:
   (a) Partially decrypt until $X_0^6[0]$ and compute the sum over all states.
   (b) If this sum is nonzero, discard the key candidate.
   (c) Otherwise, repeat the steps with the current key candidate with $\mathcal{C}_2$. Again, discard all candidates where the output is nonzero. This process is repeated

for all $\ell \in \{1, \ldots, s\}$ and only key candidates that produce a zero sum for all s structures are kept in $\mathcal{K}$.

5. Repeat the process for different bytes of X_0^6 to recover a different set of the key space, e.g., in X_0^6 to recover the other inverse diagonals of the key slices $\widehat{K}_i^{11}$;
6. After this stage, output all key candidates that satisfied all tests.

E.1.2.2 Complexity

Each key-space test has a probability to let a fraction of $2^{-8 \cdot s}$ out of at most $2^{21 \cdot 8}$ key values survive. For $s = 22$, we expect a probability of approximately 2^{-8} for any wrong key to survive, and 2^{-6} that any wrong key survives four iteration for the four rows of the final subkey. Therefore, we expect only the correct key and very few false positives, if any, to survive. The attack has the following costs:

- $s \cdot 2^{128}$ encryptions and $2s \cdot 2^{128}$ MAs to the huge tables $\mathcal{C}_i$ for storing and reading the ciphertexts;
- $2 \cdot s \cdot 2^{128} \cdot 2^k$ read accesses into $\mathcal{C}_i$;
- $2 \cdot s \cdot 2^{128} \cdot 2^k$ partial encryptions of 21 out of 256 S-boxes and additions, that are at most $2s \cdot 2^{21 \cdot 8 + 128} \cdot 21/256 \approx s \cdot 2^{293.4}$ EEs; and
- A negligible amount of surviving keys to store in $\mathcal{K}$.

Thus, the data-collection phase needs $22 \cdot 2^{128} \approx 2^{132.5}$ EEs and $2^{133.5}$ MAs. The attack needs $2 \cdot 22 \cdot 2^{128+21 \cdot 8} \cdot 21/256 \approx 2^{297.9}$ EEs and $2 \cdot 22 \cdot 2^{128+21 \cdot 8} \approx 2^{301.5}$ MAs. The memory is dominated by storing 2^{128} texts at a time.

- Data-collection phase: $2^{132.5}$ EEs, $2^{133.5}$ MAs, 2^{128} CPs.
- Memory: 2^{128} states.
- Attack phase: $2^{297.9}$ EEs and $2^{301.5}$ MAs to large tables.

Assuming that each memory access has a cost comparable to a full 10-round encryption, we obtain an attack complexity of at most $2^{301.6}$ EEs.

E.1.2.3 Using the FFT

We briefly describe how to improve the attack using the FFT technique.

We have the same separation of the involved key material into $k = k_1 + k_2$ bits, where k_1 denotes the bits directly before the final S-box and k_2 the remaining involved bits, with again $2^t = 2^{128}$ plaintext values, $k_1 = 8$ bits for the byte in $\widehat{K}^9$ and $k_2 = 160$ for the remaining 20 bytes of guessed keys. The four steps are

- A circuit-evaluation step that creates a $2^{k_1} \cdot 2^{k_2}$-dimensional vector with $c_1 = 2^{k_1} \cdot k_1 \cdot k_2 \cdot 2^{k_2}$ additions. This is $c_1 = 2^8 \cdot 8 \cdot 160 \cdot 2^{160} \approx 2^{178.3}$ additions in our case.
- A ciphertext-evaluation step of $c_2 = t \cdot 2^t$ additions. This yields $c_2 = 2^{135}$ additions in our case.
- A key-recovery step of $2^{k_1} \cdot k_1 \cdot 2^{k_2}$ multiplications, or $2^{k_1} \cdot k_1 \cdot 2^{k_2} \cdot 2^5$ additions, plus $2^{k_1} \cdot k_1 \cdot k_2 \cdot 2^{k_2}$ additions.
- Here, we obtain $c_3 = (2^8 \cdot 8 \cdot 2^{160}) \cdot (2^5 + 160) \approx 2^{178.6}$ additions.

The final effort must then be repeated s times to filter the keys, and four times in our case for the four rows of the final subkey to obtain candidates for the full key, which yields $4 \cdot (2^{178.3} + 22 \cdot 2^{178.6}) \approx 2^{185.1}$ additions. Note that those approaches usually neglect MAs that can be expensive in our case. Therefore, we expect approximately the same number of read accesses to $\mathcal{C}$, and refrain from approximating the number of additions with EEs.

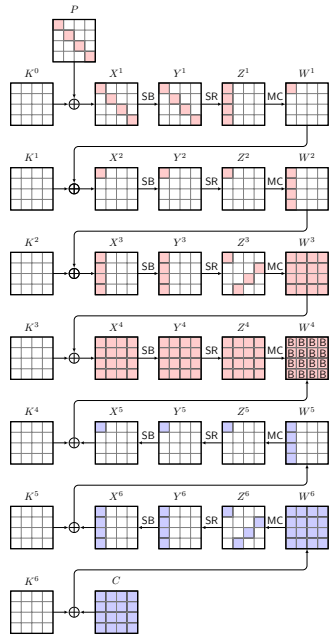


Figure 16: Integral attack on six-round AES-128.

E.1.3 Integral Attack on 12-round Vistrutah-512

In the following, we can mount a variant of the well-known integral attack on six-round AES that is shown in Figure 16 on 12-round Vistrutah-512.

E.1.3.1 Distinguisher

We can construct either a seven- or an eight-round integral trail for Vistrutah-512, where the latter is possible only when starting from an odd number of rounds, i.e., inside a step. In the following, we construct such an eight-round integral trail as shown in Figure 17, covering Rounds 2–9. The trail can be seen to propagate from any four-byte column in W_i^4; A structure that iterates over all 2^{128} values of the i-th diagonal in all slices of X^2 will propragate to a set of states that iterate over all 2^{128} values of W_i^4 and are constant in all other states of W_j^4 for $j \neq i$. Thus, it contains 2^{96} structures of 2^{32} values each that takes on all possible values in $(GF(2^8))^4$ in $W_i^4[0,1,2,3]$ and is constant in all other state bytes. Such a structure itself takes all values of $X_0^5[0,4,8,12]$ and all values in W_0^5. From the permutation property of MixColumns, each set of 2^8 values in a byte of X_0^5 is mapped to the set of all 2^8 values in a column on W_0^5. Hence, the structure of 2^{32} values of $X_0^5[0,4,8,12]$ produces a structure where every diagonal of W_0^5 and therefore every column of W_0^6 takes on all 2^{32} values. Thereupon, each column of W_0^6 is distributed to a different state slice, and a similar propagation follows to W_0^8 and to the rows of X_i^9 and until Z_i^9 for all $i \in \{0,1,2,3\}$. That is, each row of Z_i^9 iterates over all 2^{32} values in the structure, and leads to a balanced state in all bytes of W_i^9. Thus, the eight-round distinguisher from the structure of 2^{128} values in all i-th diagonals of X^2 produces a zero sum at any byte of W^9. Again, one could extend the distinguisher by one more round in front; however, this would require the entire codebook and yield no distinguisher anymore.

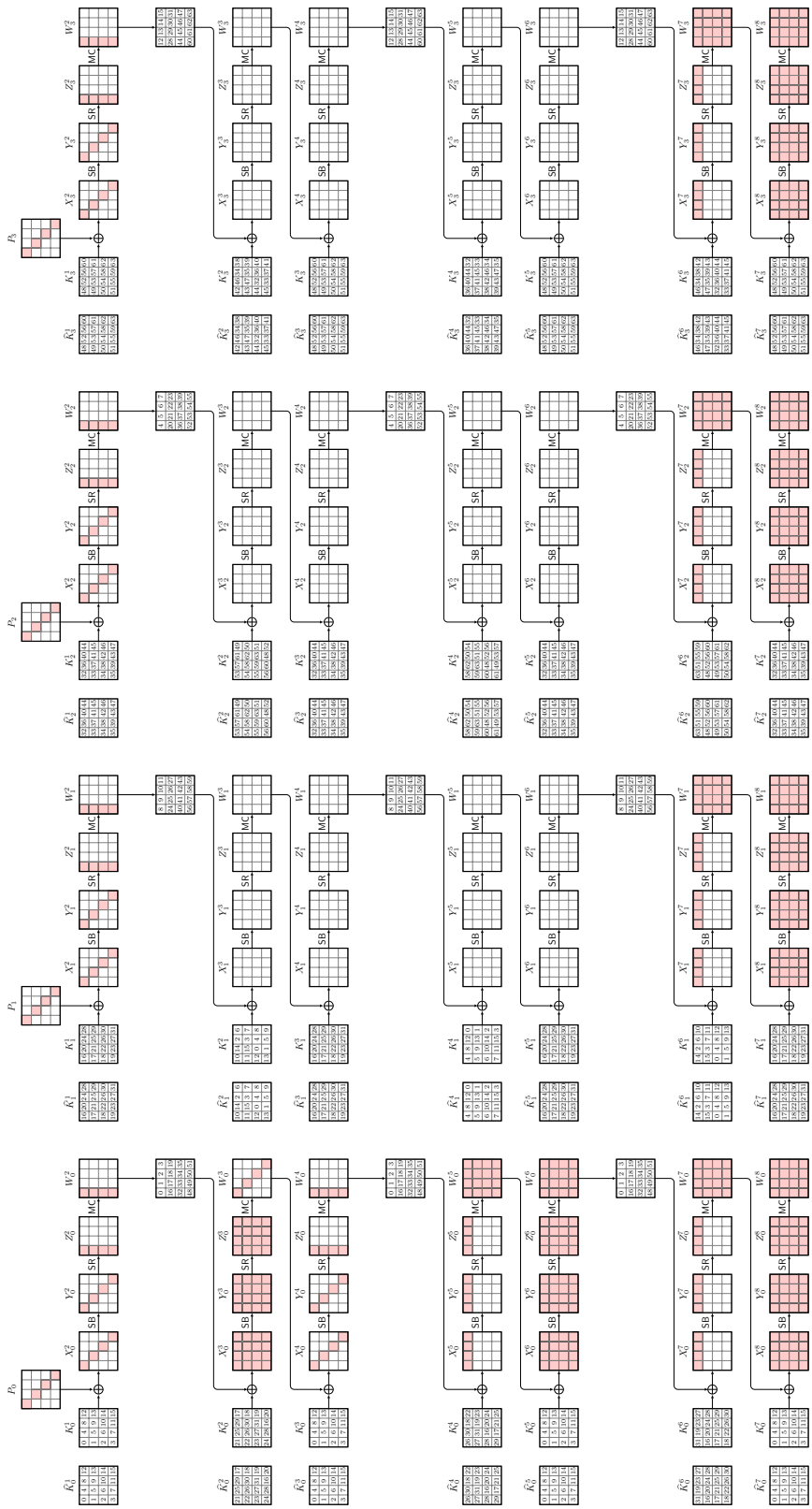


Figure 17: Integral trail on eight-round `Vistrutah`-512.

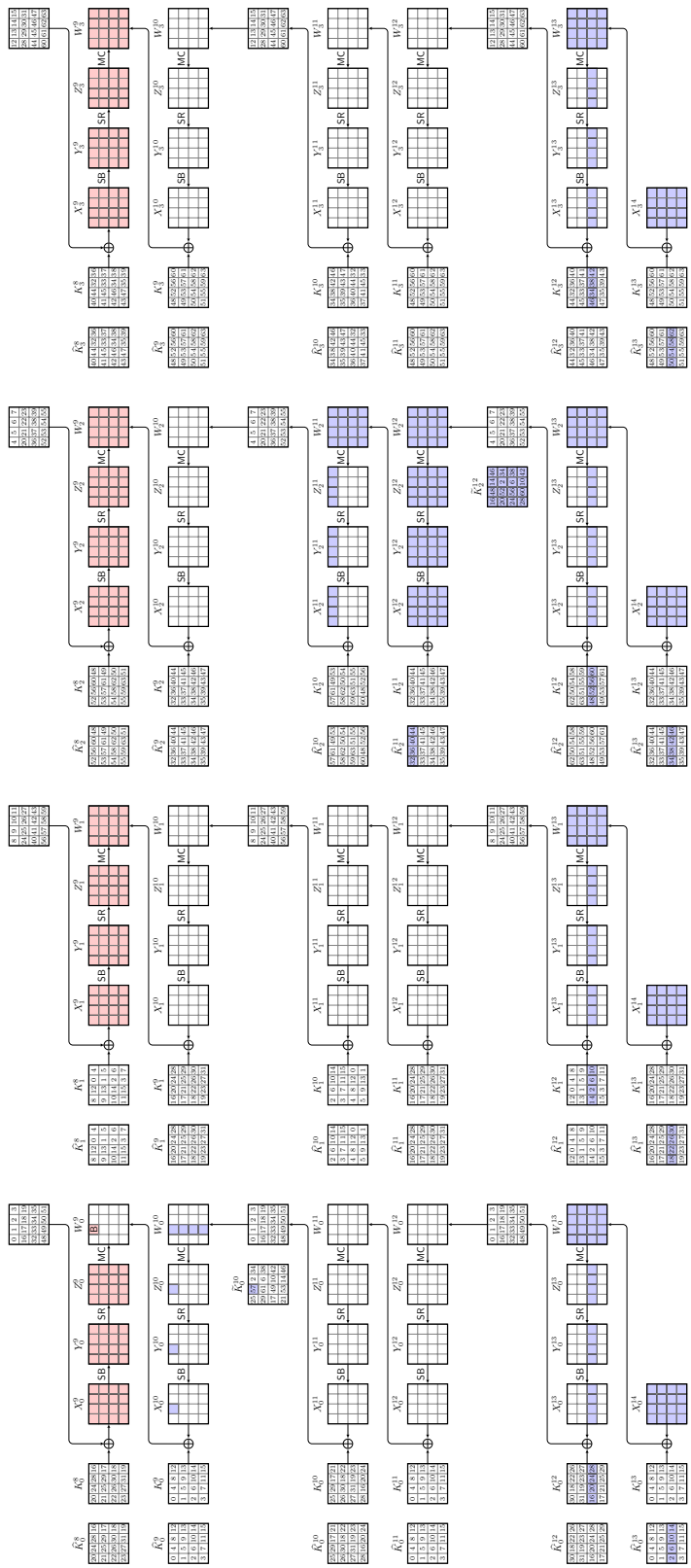


Figure 18: The key-recovery rounds of the 12-round integral attack on Vistrutah-512.

E.1.3.2 Key Recovery

The final round of the distinguisher and an example for a key-recovery trail is depicted in Figure 18. It appends four rounds, covering Rounds 9–12.

For round i and slice $j \in \{0,1,2,3\}$, we define $\widetilde{K}_j^i$ as the key K_j^i with the inverse `MixColumns` operation applied to the *rows* of $\widetilde{K}_j^i$ (and not to the columns), i.e. $\widetilde{K}_j^i = (\mathsf{MC}^{-1}((K_j^i)^\top))^\top$.

During the attack, We stepwise test and verify the correctness for different parts of the key space; Figure 18 depicts one such part. The 37 involved key bytes of the setting therein are:

- $\widetilde{K}_0^{10}[4]$,
- $\widehat{K}_2^{11}[0,4,8,12]$,
- $K_i^{12}[2,6,10,14]$, and
- $\widehat{K}_i^{13}[2,6,10,14]$ for all $i \in \{0,1,2,3\}$.

Since the information from the 16 bytes of K_i^{12} yield eight bytes of the keys $\widehat{K}_i^{13}$ corresponding to the secret-key bytes $SK[2,6,10,14,34,38,42,46]$, the number of key bytes reduces to 29.

E.1.3.3 Attack

In the following, we outline a naive attack whose time complexity can be reduced later with the FFT and partial-sum techniques. The attack consists of the following steps:

1. Initialize an empty list of key candidates $\mathcal{K}$.
2. Construct s structures of 2^{128} structures $\mathcal{P}$ each that iterate over all 16 bytes of the first diagonals of all plaintext slices P_i.
3. Ask for their corresponding encryptions after Round 13, invert the final `MixColumns` operation, and store the states into a list $\mathcal{C}_\ell$ for $\ell \in \{1,\ldots,s\}$.
4. For all key candidates and all states in $\mathcal{C}_1$:
   (a) Partially decrypt the ciphertext until $X_0^{10}[4]$ and compute the sum over all states.
   (b) If this sum is nonzero, discard the key candidate.
   (c) Otherwise, repeat the same procedure for the current key candidate with $\mathcal{C}_2$. Again, all candidates where the output is nonzero are discarded. This process is repeated for all $\ell \in \{1,\ldots,s\}$ and only key candidates that produce a zero sum for all s structures are stored into $\mathcal{K}$.
5. Repeat the process for the three different bytes of W_0^{10} to recover three further key bytes of $\widetilde{K}_0^{10}$.
6. Next, repeat the process four times to recover the four bytes of $W_1^{10}[4..7]$, then with $W_2^{10}[4..7]$ and $W_3^{10}[4..7]$. In the course, recover K_2^{10} and $\widehat{K}_2^{11}$ step by step, corresponding to $SK[32..63]$. Together with the key bytes $SK[2, 6, 10, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32]$, we know 44 key bytes, and can test the 20 remaining bytes with one encryption each.
7. Output all key candidates that satisfied all tests.

E.1.3.4 Complexity

Since each key-space test has a probability of $2^{-8\cdot s}$, at most $2^{29\cdot 8}$ key values survive. For $s = 32$, we expect a probability of approximately 2^{-24} that any wrong key survives. Since we repeat the experiment for 16 times for all 16 bytes in the second columns in the state

slices W_i^{10}, we expect a probability 2^{-20} that a wrong key survives. The attack has the following costs:

- $s \cdot 2^{128} = 2^{133}$ encryptions and $2s \cdot 2^{128} = 2^{134}$ MAs to the huge table $\mathcal{C}$ for storing and reading the ciphertexts,
- $s \cdot 2^{128} \cdot 2^{29 \cdot 8}$ partial encryptions of $1 + 4 + 16 + 16 = 37$ out of $12 \cdot 64 = 768$ S-boxes and additions, yielding $32 \cdot 2^{128+29 \cdot 8} \cdot 37/768 \approx 2^{360.6}$ encryptions.
- Each of the 15 further updates of bytes changes only one or five bytes, and iterates only over $15 \cdot s \cdot 2^{128} \cdot 2^{5 \cdot 8} \cdot 37/768 \approx 2^{172.5}$ encryptions together.
- A test with $2^{20 \cdot 8} = 2^{160}$ encryptions on average each.

Thus, the data-collection phase needs 2^{133} EEs and $2^{133.5}$ MAs. The attack needs $2^{360.6} + 2^{172.5} + 2^{160} \approx 2^{360.5}$ EEs and $16 \cdot 32 \cdot 2^{29 \cdot 8+128} \approx 2^{364.6}$ MAs. The memory is dominated by the storage of 2^{128} texts at a time.

- Data-collection phase: 2^{133} EEs, 2^{134} MAs, 2^{133} CPs.
- Memory: 2^{128} states.
- Attack phase: $2^{360.6}$ EEs and $2^{364.6}$ MAs to large tables.

Assuming that each memory access is as expensive as a full 12-round encryption, we obtain an attack complexity of at most $2^{364.7}$ EEs.

E.1.3.5 Using the FFT

Instead of computing $2^t \cdot 2^k$ sums, we again separate the involved key material into $k = k_1 + k_2$ bits. For our case, we have $2^t = 2^{128}$ plaintext values, $k_1 = 8$ bits for the byte in $\widehat{K}^9$ and $k_2 = 224$ for the remaining 28 bytes of guessed keys. The four steps are:

- A circuit-evaluation step that creates a $2^{k_1} \cdot 2^{k_2}$-dimensional vector with $c_1 = 2^{k_1} \cdot k_1 \cdot k_2 \cdot 2^{k_2}$ additions, i.e., $c_1 = 2^8 \cdot 8 \cdot 224 \cdot 2^{224} \approx 2^{242.8}$ additions in our case.
- A ciphertext-evaluation step of $c_2 = t \cdot 2^t$ additions, where $c_2 = 2^{135}$ in our case.
- A key-recovery step of $2^{k_1} \cdot k_1 \cdot 2^{k_2}$ multiplications, or $2^{k_1} \cdot k_1 \cdot 2^{k_2} \cdot 2^5$ additions, plus $2^{k_1} \cdot k_1 \cdot k_2 \cdot 2^{k_2}$ additions. This is $c_3 = (2^8 \cdot 8 \cdot 2^{224}) \cdot (2^5 + 224) \approx 2^{243.9}$ additions.

The final effort must then be repeated s times to filter the keys, and 16 times in our case to obtain 44 key bytes, plus 2^{160} further encryptions on average, which yields $16 \cdot (2^{242.8} + 32 \cdot 2^{243.9}) \approx 2^{252.9}$ additions. Note that those approaches usually neglect MAs that can be expensive in our case. Therefore, we expect approximately the same number of read accesses to $\mathcal{C}$, and choose to not approximate the number of additions with EEs.

E.1.4 Integral Attack on 12-round Vistrutah-512-256

A close variant of the integral attack on six-step Vistrutah-512 can be transferred also to Vistrutah-512-256. We use the same distinguisher from Figure 17 but have to consider the different key schedule and remove the MixColumns in the final round.

E.1.4.1 Key Recovery

The final round of the distinguisher and a key-recovery trail is depicted in Figure 19. It appends almost the same four rounds, covering Rounds 9–12, but guesses a different subset of key bytes. For round i and slice $j \in \{0, 1, 2, 3\}$, we define $\widetilde{K}_j^i$ as the key K_j^i with the inverse MixColumns operation applied to the *rows* of $\widetilde{K}_j^i$ (and not to the columns), i.e. $\widetilde{K}_j^i = (\mathsf{MC}^{-1}((K_j^i)^\top))^\top$.

During the attack, we will stepwise test and verify the correctness for different parts of the key space; Figure 18 depicts one such part. The 37 involved key bytes of the setting therein are

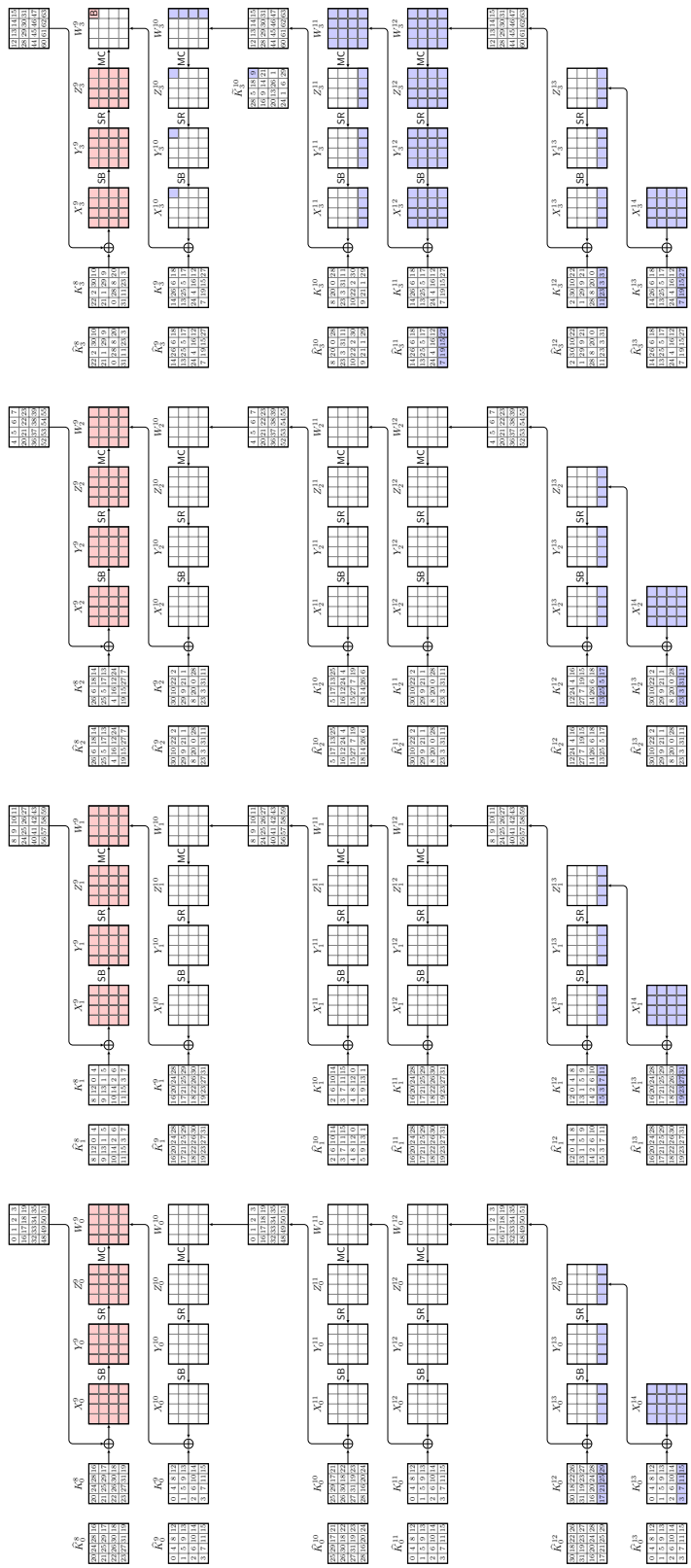


Figure 19: The key-recovery rounds of the 12-round integral attack on `Vistrutah`-512.

- $\widetilde{K}_3^{10}[12]$,
- $\widehat{K}_3^{11}[3,7,11,15]$,
- $K_i^{12}[3,7,11,15]$, and
- $K_i^{13}[3,7,11,15]$ for all $i \in \{0,1,2,3\}$.

The information from the eight bytes of $K_i^{13}[3,7,11,15]$, $i \in \{0,1\}$ is mirrored in the eight bytes of $K_j^{13}[3,7,11,15]$, $j \in \{2,3\}$. Moreover, without the final `MixColumns` operation, eight of those bytes are contained also in the rows of $K_i^{12}[3,7,11,15]$ for $i \in \{0,1,2,3\}$. In total, the key recovery involves $\widetilde{K}_3^{10}[12]$ (which needs $SK[1,21]$ not contained in the rest), $\widehat{K}_3^{11}[3,7,11,15]$, and $SK[3, 5, 7, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31]$, i.e., 19 key bytes.

E.1.4.2 Attack

We just outline the attack. The attack consists the following steps:

1. Initialize an empty list of key candidates $\mathcal{K}$.
2. Construct s structures of 2^{128} structures $\mathcal{P}$ each that iterate over all 16 bytes of the first diagonals of all plaintext slices P_i.
3. Ask for their corresponding encryptions after Round 13, invert the final `MixColumns` operation, and store the states into a list $\mathcal{C}_\ell$ for $\ell \in \{1,\ldots,s\}$.
4. For all key candidates and all states in $\mathcal{C}_1$:
   (a) Partially decrypt the state to $X_3^{10}[12]$ and compute the sum thereof over all states.
   (b) If the sum is nonzero, discard the key candidate.
   (c) Otherwise, repeat the same procedure for the current key candidate and with the states with $\mathcal{C}_2$. Discard all candidates where the output is nonzero.
   (d) Repeat for all $\ell \in \{1,\ldots,s\}$ and store the key candidates that produce a zero sum for all s structures into $\mathcal{K}$.
5. As a result, we obtain 19 key bytes.
6. Guess and test the 13 missing key bytes with one encryption per key candidate.
7. Output the key candidates that satisfy all tests.

E.1.4.3 Complexity

Each test has a probability to let a fraction of $2^{-8\cdot s}$ out of at most $2^{19\cdot 8}$ key values survive. For $s = 20$, we expect a probability of approximately 2^{-8} that any wrong key survives. Since we repeat the experiment for 16 times for all 16 bytes in the second columns in the state slices W_i^{10}, we expect a probability 2^{-4} that a wrong key survives. The attack has the following costs:

- $s \cdot 2^{128} = 2^{132.3}$ encryptions and $2s \cdot 2^{128} = 2^{133.3}$ MAs to the huge tables $\mathcal{C}_\ell$ for storing and reading the ciphertexts,
- $s \cdot 2^{128} \cdot 2^{19\cdot 8}$ partial encryptions of $1+4+16+16 = 37$ out of $12 \cdot 64 = 768$ S-boxes and additions, yielding $20 \cdot 2^{128+19\cdot 8} \cdot 37/768 \approx 2^{279.9}$ encryptions.
- A test with $2^{13\cdot 8} = 2^{104}$ encryptions on average each.

Thus, the data-collection phase needs $2^{132.3}$ EEs and $2^{133.3}$ MAs. The attack needs $2^{279.9} + 2^{104} \approx 2^{279.9}$ EEs and $16 \cdot 20 \cdot 2^{19\cdot 8+128} \approx 2^{288.3}$ MAs. The memory complexity is dominated by storage of 2^{128} texts at a time.

- Data-collection phase: $2^{132.3}$ EEs, $2^{133.3}$ MAs, $2^{132.3}$ CPs.
- Memory: 2^{128} states.
- Attack phase: $2^{279.9}$ EEs and $2^{288.3}$ MAs to large tables.

Assuming that each memory access is as expensive as a full 12-round encryption, we obtain an attack complexity of at most $2^{288.3}$ EEs.

E.1.4.4 Using the FFT

Again, we can use the FFT to improve the attack:

- A circuit-evaluation step that creates a $2^{k_1} \cdot 2^{k_2}$-dimensional vector with $c_1 = 2^{k_1} \cdot k_1 \cdot k_2 \cdot 2^{k_2}$ additions. It is $c_1 = 2^8 \cdot 8 \cdot 144 \cdot 2^{144} \approx 2^{162.2}$ in our case.
- A ciphertext-evaluation step of $c_2 = t \cdot 2^t$ additions. It is $c_2 = 2^{135}$ in our case.
- A key-recovery step of $2^{k_1} \cdot k_1 \cdot 2^{k_2}$ multiplications, or $2^{k_1} \cdot k_1 \cdot 2^{k_2} \cdot 20$ additions, plus $2^{k_1} \cdot k_1 \cdot k_2 \cdot 2^{k_2}$ additions. It is $c_3 = (2^8 \cdot 8 \cdot 2^{144}) \cdot (20 + 144) \approx 2^{162.4}$ in our case.

The final effort must then be repeated s times to filter the keys plus 2^{104} further encryptions on average, which yields $2^{162.2} + 20 \cdot 2^{162.4} \approx 2^{166.8}$ additions. Since FFT approaches usually neglect MAs to huge tables, we expect roughly the same number of read accesses to the tables $\mathcal{C}_\ell$. Hence, we choose to not approximate the number of additions with EEs.

E.2 Impossible-differential Cryptanalysis

E.2.1 Impossible-differential Attack on 7-round `Vistrutah`-256

E.2.1.1 Trails

An impossible differential (ID) trail is shown in Figure 20, covering Rounds 3 through 7. The 12 bytes from SK_0, $SK[0, 2..5, 7..10, 13..15]$ and three bytes from SK_1, $SK[16, 21, 26]$ are the involved keys at the top. The figure shows one out of eight trails that can be used for the same key bytes:

- two options for ΔW_1^2, whether only $\Delta W_1^2[0, 2]$ or $\Delta W_1^2[1, 3]$ are active,
- four options for the same anti-diagonal being inactive in each slice of ΔZ^7.

For the full attack, we will also use three other variants of the trail, considering the intersection of the leftmost three columns with the three other diagonals in K_1^1.

E.2.1.2 Attack

The attack consists of the following steps:

1. Precompute lookup tables $\mathcal{H}_1, \ldots, \mathcal{H}_{11}$ as follows:
   - Given a 32-bit input difference, $\mathcal{H}_1$, $\mathcal{H}_2$, and $\mathcal{H}_3$ return all values after the key addition for one of the texts in a pair with that difference that have a single active byte after one round of a Super-box at index 0, 1, 2 for $\mathcal{H}_1$, $\mathcal{H}_2$, and $\mathcal{H}_3$, respectively.
   - Given a 24-bit input difference, $\mathcal{H}_4$ and $\mathcal{H}_5$ return all values after the key addition for one of the texts in a pair with that 3-byte difference in bytes $(0, 1, 2)$ that have two active bytes after one round of a Super-box at indices $(0, 2)$ and $(1, 3)$ for $\mathcal{H}_4$ and $\mathcal{H}_5$, respectively.
   - The remaining table pairs have different three active input bytes: Bytes $(0, 1, 3)$ for $\mathcal{H}_6$ and $\mathcal{H}_7$, Bytes $(0, 2, 3)$ for $\mathcal{H}_8$ and $\mathcal{H}_9$, and Bytes $(1, 2, 3)$ for $\mathcal{H}_{10}$ and $\mathcal{H}_{11}$ to address the other diagonals in K_1^1.
2. Initialize four lists $\mathcal{K}_i$ of $2^{15} \cdot 80$-bits each, for the keys $SK[0, 2..5, 7..10, 13..15]$ and $SK[16, 21, 26]$, $SK[19, 20, 25]$, $[18, 23, 24]$, and $[17, 22, 27]$, respectively.
3. Initialize four empty hash tables of ciphertexts $\mathcal{C}_i$.
4. Build a structure of 2^s random distinct plaintexts differing in the first three diagonals of P_1 and leaving all other plaintext diagonals constant.

5. Ask for their corresponding encryptions C, invert the final `MixColumns` operation and store all texts into $\mathcal{C}_i$ indexed by the values of the i-th anti-diagonals in both slices, concatenated.
6. Identify pairs at the same positions in the tables.
7. For each of the pairs identified in Step 6:
   (a) Guess the 32-bit differences in $\Delta W_1^1[0, 5, 10]$ and $\Delta W_1^2[0]$ (or $\Delta W^2[1]$).
   (b) For each guess, derive the corresponding one key on average that produces that difference with the precomputed lookup tables and set its bit in $\mathcal{K}$ to one.
   (c) Repeat the procedure of all pairs with the three diagonals of ΔW_1^1 and two options in ΔW_1^2 each.
8. Discard all keys with 1 in $\mathcal{K}$ as impossible.
9. For all surviving keys, build their set of overlapping keys.
10. Guess the remaining eight key bytes:
    (a) Output the candidate keys that pass a full encryption test.

E.2.1.3 Complexity

The attack involves 15 key bytes, i.e., 120 bits. The probability for each ciphertext pair to have the same anti-diagonal inactive in both slices of $\Delta\widehat{W}^7$ is approximately $4 \cdot 2^{-64} = 2^{-62}$. Since we can build 2^{2s-1} pairs from 2^s texts, we expect 2^{2s-63} pairs to survive this first

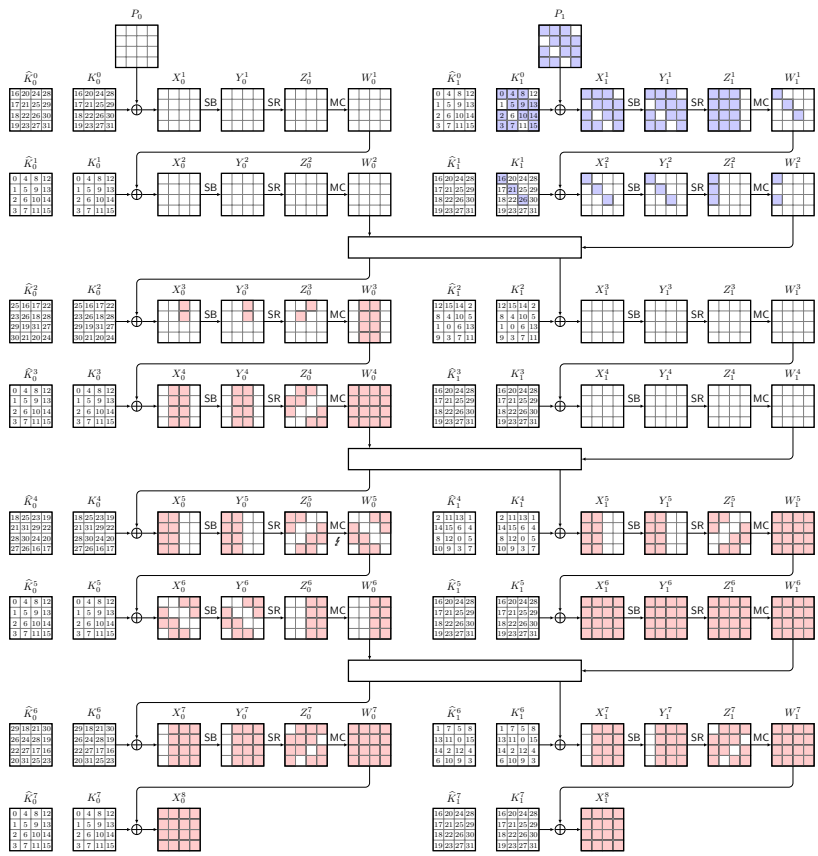


Figure 20: Distinguisher and key-recovery phases in the 7-round impossible-differential attack on `Vistrutah`-256.

filter. Each surviving pair suggests $2 \cdot 2^{32} = 2^{33}$ keys that would lead to an impossible differential on average for a fixed diagonal in K_1^1. Thus, the probability that one fixed key is marked as impossible is approximately $2^{120-33} = 2^{87}$. To have $(1 - 2^{-87})^N$ sufficiently small, we need approximately $N = 1/2^{-87-7} = 2^{94}$ pairs surviving the initial ciphertext-pair filter. Then, the probability that any wrong 120-bit key candidate survives the filter is approximately $2^{120} \cdot 2^{-184.6} \approx 2^{-64.6}$ so that we expect only the correct key and a very few more to survive. This number demands that we have $2^{94+62} = 2^{156}$ pairs, which corresponds to $2^s = 2^{78.5}$; thus, 2^{79} CPs should allow to form the necessary number of pairs. The complexity consists of:

- $(3 \cdot 2^8 + 8 \cdot 2^{16}) \cdot 2 \cdot 2^{32} \cdot 4/(7 \cdot 32) \approx 2^{46.2}$ seven-round EEs for precomputations,
- Encrypting 2^{79} plaintexts,
- Writing them into and reading them into four tables $\mathcal{C}_i$ with $4 \cdot 2 \cdot 2^{79} = 2^{82}$ MAs,
- Extracting ca. $2^{157-62} \approx 2^{95}$ pairs with 2^{95} MAs,
- Extracting the corresponding keys with $2^{32} \cdot 8 = 2^{35}$ MAs per pair,
- Setting $4 \cdot 2^{33}$ flags on average in $\mathcal{K}$,
- Extracting all surviving key candidates with 2^{120} MAs, and
- Testing the surviving keys with 2^{64} further encryptions each.

Therefore, the encryption costs consist of $2^{46.2} + 2^{79} + 2^{64} \approx 2^{79}$ EEs and $2^{82} + 2^{95} + 2^{95} \cdot (2^{35} + 4 \cdot 2^{33}) + 2^{120} \approx 2^{131}$ MAs to potentially large tables that we approximate with at least one eight-round encryption each. The attack needs memory for 2^{120} bits of key candidates plus $4 \cdot 2^{79}$ plaintext-ciphertext pairs at a time plus the negligible precomputation tables, i.e., $2^{120} \cdot 1/256 + 2^{81} \cdot 2 \approx 2^{112}$ states. In total, the attack complexities are approximately

- Data-collection phase: 2^{79} EEs, 2^{82} MAs, 2^{79} CPs.
- Memory: 2^{112} states.
- Attack phase: 2^{131} MAs to large tables.

E.2.2 Impossible-differential Attack on 8-round `Vistrutah`-256

E.2.2.1 Trails

An impossible differential trail is shown in Figure 21, covering Rounds 3 through 6. Many related and structurally rotated trails are possible. For the full attack, we consider also a mirrored trail version with the wrapping rounds being active on the lefthand side of the figure, i.e., with ΔP_0 and ΔC_0 being active. The involved keys at the top are 12 bytes from SK_0, $SK[0, 2..5, 7..10, 13..15]$ and three bytes from SK_1, $SK[16, 21, 26]$. Moreover, the key-guessing phase at the end considers the final key K_1^8, which corresponds to $SK[0..15]$. There are $2 \cdot \binom{4}{2} = 12$ options of pairs of two-byte index pairs in ΔX_1^7 that can be inactive to obtain an impossible differential: choosing any two out of $\{(0,1), (2,3), (4,5), (6,7)\}$ and similarly, choosing any two out of $\{(8,9), (10,11), (12,13), (14,15)\}$.

E.2.2.2 Attack

The attack consists of the following steps:

1. Precompute lookup tables $\mathcal{H}_1$, $\mathcal{H}_2$, and $\mathcal{H}_3$.
2. Given a four byte input difference $(\alpha_0, \alpha_1, \alpha_2, \alpha_3) \in (\mathbb{F}_{2^8})^4$, $\mathcal{H}_1[(\alpha_0, \alpha_1, \alpha_2, \alpha_3)]$ contains exactly the approximately 2^8 states on average that lead to differences $(\beta_0, 0, 0, 0)$ for some $\beta_0 \in \mathbb{F}_{2^8}$ after one round of `AES` (for one column).
3. $\mathcal{H}_2[(\alpha_0, \alpha_1, \alpha_2, 0)]$ contains all 2^8 states on average that lead to $(\beta_0, 0, \beta_1, 0) \in (\mathbb{F}_{2^8})^4$ for some β_0, β_1 after one round.
4. $\mathcal{H}_3[(\alpha_0, \alpha_1, \alpha_2, 0)]$ contains all 2^8 states on average that lead to $(0, \beta_1, \beta_2, \beta_3) \in (\mathbb{F}_{2^8})^4$ for some $\beta_1, \beta_2, \beta_3$ after applying inverse `SubBytes` and inverse `MixColumns`.

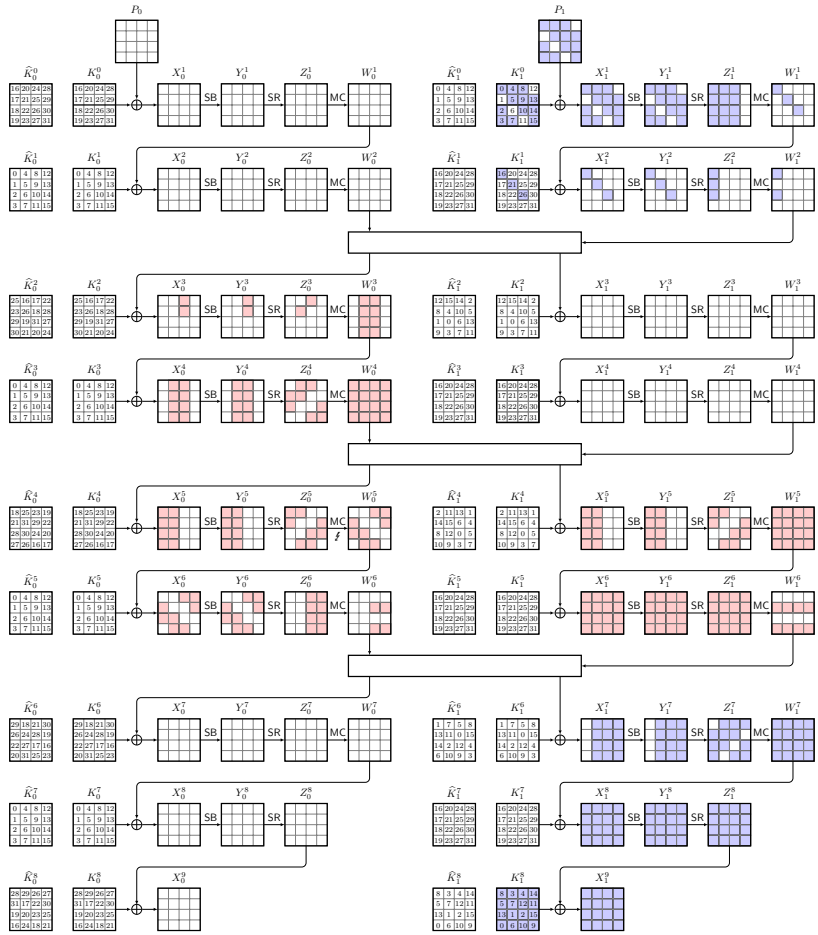


Figure 21: Distinguisher and key-recovery phases in the 8-round impossible-differential attack on Vistrutah-256.

5. Initialize a list $\mathcal{K}$ of $2^{19.8}$ bits for the involved keys $SK[0..16, 21, 26]$, with all entries initially all set to zero.
6. Initialize an empty hash table of ciphertexts $\mathcal{C}$.
7. Choose 2^s structures of 2^{96} plaintexts that iterate over all values in the first three diagonals of P_1 and having all other bytes assigned to random values that are the same for all plaintexts.
8. Ask for their corresponding encryptions C and store them into $\mathcal{C}$ indexed by their left parts C_0.
9. Identify the pairs that share the same values for C_0.
10. For each pair identified in 9:
    (a) Guess the 32-bit differences in $\Delta W_1^1[0, 5, 10]$, $\Delta W_1^2[0]$, and twelve options of twelve bytes of ΔY_1^7.
    (b) For each option of ΔY_1^7, derive the corresponding 128-bit keys on average that produced that difference.
    (c) For each of $12 \cdot 2^{32+96-96} \approx 2^{35.6}$ overlapping options, set their bits to 1 in $\mathcal{K}$.
11. Discard all keys with 1 in $\mathcal{K}$ as impossible.
12. Repeat the attack with another set of 2^s plaintexts but now with active $\mathcal{D}_{0,1,2}(P_0)$

in the plaintext differences and active ΔC_0 instead.
13. Keep all overlapping keys.
14. Test these keys by trial encryptions.

E.2.2.3 Complexity

The attack involves 19 key bytes, i.e., 152 bits. The probability for each ciphertext pair to have $\Delta C_0 = 0$ is approximately 2^{-128}. Each pair that survives this initial ciphertext-side filter would yield 2^{32} key candidates for $\mathcal{D}_{0,1,2}(SK_0)$ in K_1^0 and $SK[16, 21, 26]$ in the first two rounds and $12 \cdot 2^{96}$ candidates for SK_0 in the final two rounds, i.e., $2^{131.6}$ key candidates. However, the 96 bits of $\mathcal{D}_{0,1,2}(SK_0)$ from K_1^0 must match SK_0 from K_1^8, which only holds with probability 2^{-96} for a candidate. Thus, we are left with only $2^{131.6-96} = 2^{35.6}$ key candidates on average. Therefore, the probability that a random key among the 2^{152} candidates is flagged as impossible for a surviving pair is approximately $2^{152-35.6} = 2^{116.4}$. To have $(1 - 2^{-116.4})^N$ sufficiently small, we need approximately $N = 1/2^{-116.4-6} = 2^{122.4}$ pairs surviving the initial ciphertext-pair filter. This number demands that we have $2^{122.4+128} = 2^{250.4}$ pairs. Since a structure of 2^{96} texts can yield 2^{191} pairs, this implies $2^{59.4}$ structures are necessary, which corresponds to $2^{59.4+96} = 2^{155.4}$ CPs. The complexity of this procedure consists of:

- $3 \cdot 2^{32} \cdot 2^8$ precomputations through a single round of `AES`;
- Encrypting $2^{155.4}$ CPs;
- Writing them into and reading them from $\mathcal{C}$ with $2 \cdot 2^{155.4}$ MAs;
- Extracting $2^{122.4}$ pairs with $\Delta C_0 = 0$;
- Extracting the corresponding keys with 8 MAs each to $\mathcal{H}$;
- Filtering the keys to $2^{35.6}$ with 8 MAs each;
- Setting $2^{35.6}$ flags in $\mathcal{K}$;
- Extracting all surviving key candidates with 2^{152} MAs; and
- Testing the surviving keys with one encryption each.

This procedure is then repeated, this time with active differences of pairs in ΔP_0 and looking for $\Delta C_1 = 0$ instead. We expect $2^{152} \cdot (1 - 2^{-116.4})^{2^{122.4}} \approx 2^{59.7}$ candidates to survive, and a similar amount for the second phase. With the keys from the second phase, we can construct the full 32-byte key. Therefore, after the second phase, we can expect at most $2^{2 \cdot 59.7} \approx 2^{119.4}$ candidates on average that can be tested with one encryption per candidate. Therefore, the encryption costs consist of $3 \cdot 2^{32+8} + 2 \cdot 2^{155.4} + 2^{119.4} \approx 2^{156.4}$ EEs plus $2 \cdot (2^{156.4} + 2^{122.4} \cdot (8 + 8 \cdot 2^{35.6} + 2^{35.6}) + 2^{152}) \approx 2^{162.2}$ MAs to large tables that we approximate with at least one eight-round encryption each. The attack needs memory for 2^{152} bits of key candidates plus 2^{96} plaintext-ciphertext pairs for a structure at a time, i.e., $2^{144} + 2^{97} \approx 2^{144}$ states. In total, the attack needs approximately:

- Data-collection phase: $2^{156.4}$ EEs, $2^{157.4}$ MAs, $2^{156.4}$ CPs.
- Memory: 2^{144} states.
- Attack phase: $2^{162.2}$ MAs to large tables.

E.2.3 Impossible-differential Attack on 9-round `Vistrutah-256`

In the following, we mount an impossible differential attack on nine rounds, without the `MixColumns` operation in the final round.

E.2.3.1 Trails

An impossible-differential trail is shown in Figure 22, covering Rounds 3 through 8.

The involved key bytes are:

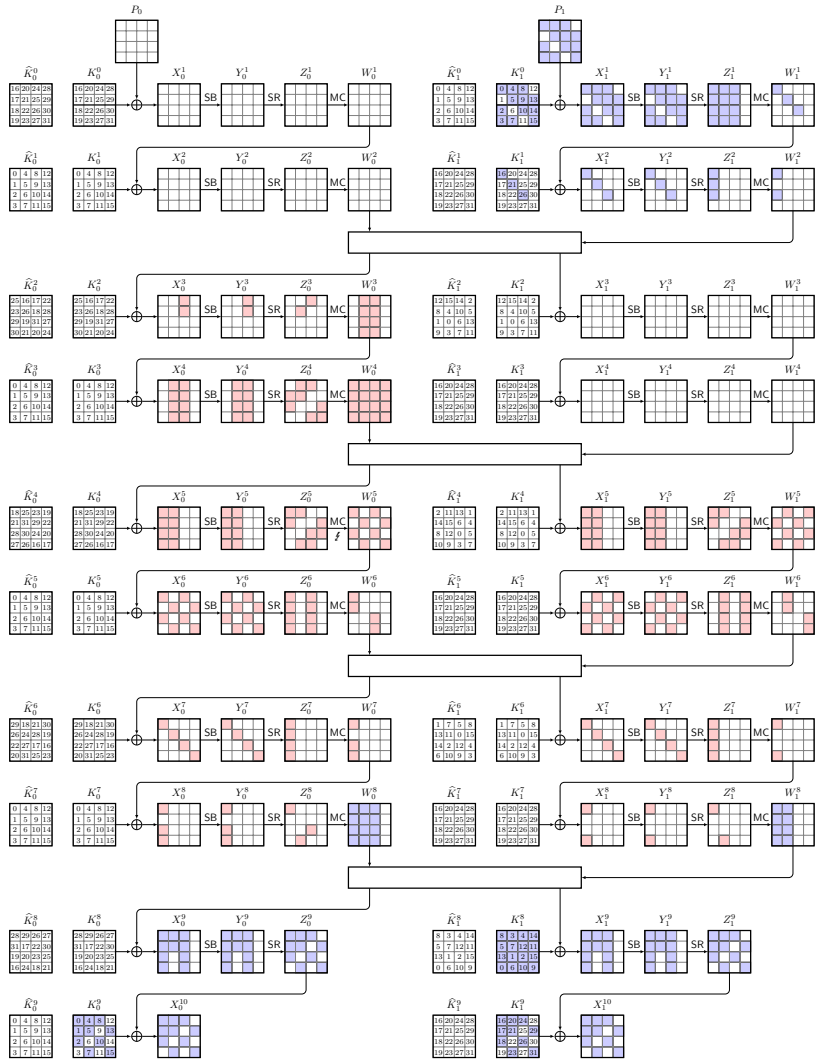


Figure 22: Distinguisher and key-recovery phases in the 8-round impossible-differential attack on Vistrutah-256.

- $K_1^0[0, 2..5, 7..10, 13..15]$ and $K_1^1[0, 5, 10]$ in the top rounds, corresponding to $SK[0..5, 7..10, 13..15]$ and $SK[16, 21, 26]$, and
- $K_0^9[0..2, 4, 5, 7, 8, 10, 13, 15]$ and $K_1^9[0..2, 4, 5, 7, 8, 10, 13, 15]$ in the final round, corresponding to the cells $SK[0..2, 4, 5, 7, 8, 10, 13, 15]$ and $SK[16..18, 20, 21, 23, 24, 26, 29, 31]$.

The 23 involved key bytes are $SK[0..5, 7..10, 13..15, 16..18, 20, 21, 23, 24, 26, 29, 31]$.

The impossible differential requires only one 128-bit slice to be active after the first step, which propagates to at most two active bytes in each column at ΔZ^5. In backwards direction, the differential allows at most one inverse diagonal active in ΔZ_0^8 and ΔZ_1^8 each, i.e., before the MixColumns operation of Round 8. This leads to at most two active bytes per column in ΔW^5, and therefore to a contradiction with ΔZ^5 due to the branch number of MixColumns. To reduce the steps in the later efforts for data collection, key-guessing, and key enumeration, we consider three active columns to be active in ΔW_0^8 and two active columns in ΔW_1^8. This configuration considers in total 2^5 impossible differential trails:

- Two options to have either $\Delta W_1^2[0,2]$ or $\Delta W_1^2[1,3]$ inactive;
- The three active bytes in $\Delta Z_0^8[0..11]$ can be in any one of the four inverse diagonals; and
- The two active bytes in $\Delta Z_1^8[0..7]$ can be in any one of the four inverse diagonals.

E.2.3.2 Attack

The attack consists of the following steps:

1. Precompute lookup tables $\mathcal{H}_1$, $\mathcal{H}_2$, and $\mathcal{H}_3$:
   - Given a four byte input difference $(\alpha_0, \alpha_1, \alpha_2, \alpha_3) \in (\mathbb{F}_{2^8})^4$, $\mathcal{H}_1[(\alpha_0, \alpha_1, \alpha_2, \alpha_3)]$ contains exactly the approximately 2^8 states on average that lead to differences $(\beta_0, 0, 0, 0)$ for some $\beta_0 \in \mathbb{F}_{2^8}$ after one round of `AES` (for one column).
   - $\mathcal{H}_2[(\alpha_0, \alpha_1, \alpha_2, 0)]$ contains all 2^8 states on average that lead to $(\beta_0, 0, \beta_1, 0) \in (\mathbb{F}_{2^8})^4$ for some β_0, β_1 after one round.
   - $\mathcal{H}_3[(\alpha_0, \alpha_1, \alpha_2, 0)]$ contains all 2^8 states on average that lead to $(0, \beta_1, \beta_2, \beta_3) \in (\mathbb{F}_{2^8})^4$ for some $\beta_1, \beta_2, \beta_3$ after applying inverse SubBytes and inverse MixColumns.
2. Initialize a list $\mathcal{K}$ of $2^{23 \cdot 8}$ bits for the involved key bytes, initially all set to zero.
3. Initialize an empty hash table of ciphertexts $\mathcal{C}$.
4. Choose 2^s structures of 2^{96} plaintexts that iterate over all values in the first three diagonals of P_1 and having all other bytes assigned to random values that are the same for all plaintexts.
5. Ask for their corresponding encryptions C and store them into $\mathcal{C}$ indexed by the 12 byte values $C_i[3, 6, 9, 11, 12, 14]$ for both $i \in \{0, 1\}$.
6. Identify the pairs that share the same indices.
7. For each pair identified in Step 6:
   (a) Guess the two options of 32-bit differences in $\Delta W_1^1[0, 5, 10]$, $\Delta W_1^2[0]$, four options of 24-bit differences in three active bytes in a single anti-diagonal in ΔZ_0^8 and four options of 16-bit differences in two active bytes in a single anti-diagonal in ΔZ_1^8; Figure 22 shows one such option, here with $\Delta W_1^2[0, 2]$, $\Delta Z_0^8[0, 7, 10]$, and $\Delta Z_1^8[0, 7]$ active.
   (b) For each option derive the corresponding 23-byte keys on average that produced that difference.
   (c) Obtain 2^{32} candidates from the top rounds, with 56 key constraints in the bottom for each.
   (d) When a key candidate satisfies all these constraints, set its bit in $\mathcal{K}$.
8. Discard all keys with 1 in $\mathcal{K}$ as impossible.
9. For all nonmarked keys, guess the remaining $32 - 23 = 9$ key bytes and test each candidate with one encryption each. Output the keys that satisfy this test.

E.2.3.3 Complexity

The attack involves 23 key bytes, i.e., 184 bits. The probability for each ciphertext pair to have the twelve bytes $\Delta C_0[3, 6, 9, 11, 12, 14]$ and $\Delta C_1[3, 6, 9, 11, 12, 14]$ all zero is 2^{-96}. Each pair that survives this initial ciphertext-side filter would yield $2^5 \cdot 2^{-24} = 2^{-19}$ key candidates on average, i.e., we need on average 2^{19} surviving pairs to filter one key candidate. In order for the probability $(1 - 2^{-184})^N$ that a key is not filtered to be sufficiently small, we need approximately $N = 1/2^{-184-19-6} = 2^{209}$ pairs that survive the initial ciphertext-pair filter. This number demands that we have $2^{209+96} = 2^{305}$ pairs in

total Since a structure of 2^{96} texts can yield 2^{191} pairs, this implies 2^{114} structures will be necessary, which corresponds to $2^{96+114} = 2^{210}$ CPs. The complexity consists of:

- $3 \cdot 2^{32} \cdot 2^8$ precomputations through a single round of AES,
- Encrypting 2^{210} CPs;
- Writing them into and reading them from $\mathcal{C}$ with $2 \cdot 2^{210}$ MA;
- Extracting 2^{209} pairs that survive the first filter;
- Extracting the corresponding keys with $2^5 \cdot 9$ MAs each to $\mathcal{H}$;
- Filtering the keys to one valid key on every 2^{19} tries on average with $2^5 \cdot 9$ MAs each;
- Setting a flag in $\mathcal{K}$ once every 2^{19} tries;
- Extracting all surviving key candidates with 2^{184} MAs; and
- Testing the surviving keys with one encryption each.

We expect $2^{184} \cdot (1 - 2^{-184})^{2^{190}} \approx 2^{91.7}$ candidates to survive, and this at most $2^{91.7} \cdot 2^{72} \approx 2^{163.7}$ candidates on average that have been tested with one encryption. The other steps of the encryption costs consist of $3 \cdot 2^{32+8} + 2^{210} + 2^{163.7} \approx 2^{210}$ EEs plus $2^{211} + 2^{209} \cdot (2^5 \cdot (9+9) + 2^{5-19}) + 2^{184} \approx 2^{218.2}$ MAs to large tables that we approximate with at least one eight-round encryption each. The attack needs memory for 2^{184} bits of key candidates plus 2^{96} plaintext-ciphertext pairs for a structure at a time, i.e., $2^{184-8} + 2^{97} \approx 2^{176}$ states. In total, the attack needs approximately:

- Data-collection phase: 2^{210} EEs, 2^{211} MAs, 2^{210} CPs.
- Memory: 2^{176} states.
- Attack phase: $2^{2^{218.2}}$ MAs to large tables.

E.2.4 Impossible-differential Attack on 10-round Vistrutah-512

In the following, we mount an impossible differential attack on nine rounds, without the MixColumns operation in the final round.

E.2.4.1 Trails

An impossible-differential trail is shown in Figure 23, covering Rounds 3 through 8. The key-recovery phase is depicted in Figure 24. The involved key bytes are the first 32 bytes, i.e., $SK[0..31]$.

However, we can use at least 36 trails:

- We can use any combination of $\binom{4}{2}$ pairs of active columns of ΔW_0^2 and the same columns in ΔW_1^2 such that exactly two 128-bit slices will be active after the mixing layer in ΔX^3. Then, all those trails will lead to at most two active bytes in any column in ΔZ^5.
- Similarly, we can use any combination of $\binom{4}{2}$ pairs of active rows of ΔX_0^9 and the same rows in ΔX_1^9 such that at most two 128-bit slices will be active after the inverse mixing layer in ΔW^8. Then, all those trails will lead to at most two active bytes in any column in ΔW^5, producing a contradiction with ΔZ^5 due to the MixColumns branch number.

E.2.4.2 Attack

First, a few lookup tables must be precomputed:

- 36 tables $\mathcal{H}_{1,(i,j),(i',j')}$ for all $(i,j),(i',j') \in \{0,1,2,3\}^{2\cdot 2}$ with $i \neq j$ and $i' \neq j'$: for a four-byte input difference $(\alpha_0, \alpha_1, \alpha_2, \alpha_3)$, outputs all values that produce a column difference of $(\beta_0, \beta_1, \beta_2, \beta_3)$ after one round of AES for any $\alpha_i, \alpha_j, \beta_{i'}, \beta_{j'} \in (GF(2^8))^4$ but all other byte differences being zero.

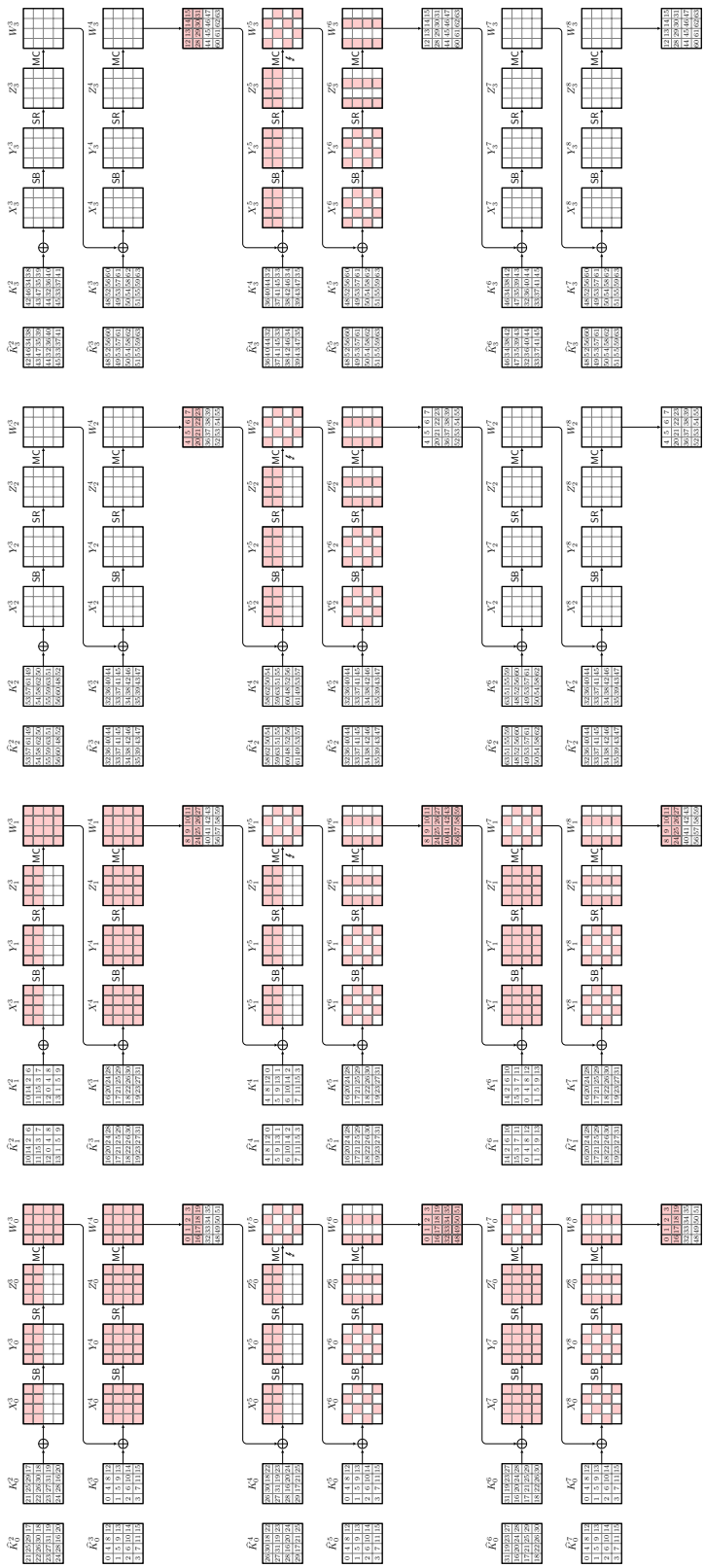


Figure 23: The impossible-differential trail in the attack on 10-round `Vistrutah`-512.

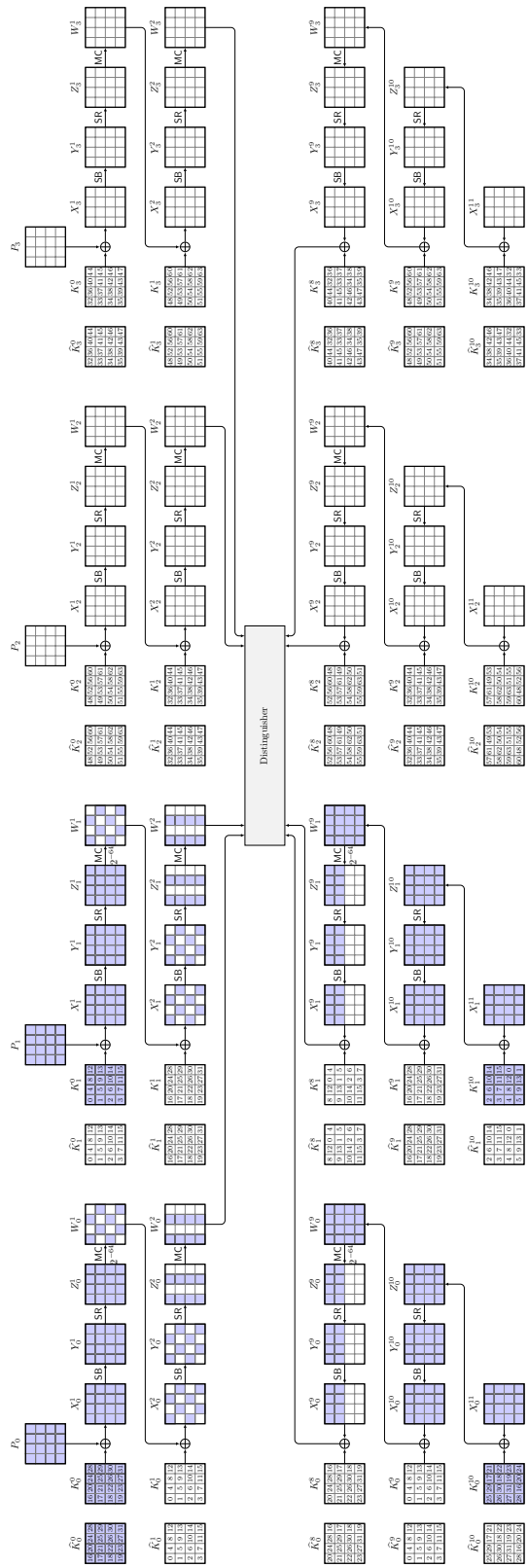


Figure 24: The key-recovery rounds of the 10-round impossible-differential attack on Vistrutah-512.

- 36 tables $\mathcal{H}_{2,(i,j),(i',j')}$ for all $(i,j),(i',j') \in \{0,1,2,3\}^{2\cdot 2}$ with $i \neq j$ and $i' \neq j'$: $(\alpha_0, \alpha_1, \alpha_2, \alpha_3)$, outputs all values that produce a column difference of $(\beta_0, \beta_1, \beta_2, \beta_3)$ after one inverse round of `AES` for any $\alpha_i, \alpha_j, \beta_{i'}, \beta_{j'} \in (GF(2^8))^4$ but all other byte differences being zero.

Then, the attack consists of the following steps:

1. Initialize a list $\mathcal{K}$ of 2^{256} 0-bits for the involved keys.
2. Initialize an empty hash table of ciphertexts $\mathcal{C}$.
3. Choose 2^s structures of 2^{256} plaintexts that iterate over all values in the first two 128-bit slices of P_0 and P_1, with all other bytes assigned to random values that are the same for all plaintexts.
4. For all plaintexts, ask for their corresponding encryptions C and store them into $\mathcal{C}$ indexed by integer representations of their 32-byte values in their ciphertext slices (C_2, C_3).
5. Identify the pairs at equal indices in $\mathcal{C}$.
6. For each such pair and for each of the 36 trail options, derive the keys that produce an impossible trail. We separate an example description for the trail in Figure 24 in the next list below.
7. For each "surviving" (here, this means leading to an impossible differential) 32-byte key option, set its bits to 1 in $\mathcal{K}$.
8. Discard all keys with 1 in $\mathcal{K}$ as impossible.
9. Output the remaining keys.

For deriving the keys that produce an impossible trail, we can do the following steps. For the sake of brevity, we only describe steps for the trail in Figure 24 as they are similar for the other trails.

1. For all 24-bit values of $SK[22, 26, 31]$:
   (a) Lookup $SK[26, 31]$ in $\mathcal{H}_{1,(2,3),(0,2)}$ to obtain $SK[16, 21]$.
   (b) Lookup $SK[21, 31]$ in $\mathcal{H}_{2,(1,3),(0,1)}$ to obtain $SK[18, 28]$.
   (c) Lookup $SK[28, 22]$ in $\mathcal{H}_{1,(0,2),(1,3)}$ to obtain $SK[17, 27]$.
   (d) Lookup $SK[22, 16]$ in $\mathcal{H}_{1,(1,3),(0,1)}$ to obtain $SK[19, 25]$.
   (e) Lookup $SK[17, 27]$ in $\mathcal{H}_{2,(0,2),(0,1)}$ to obtain $SK[24, 30]$.
   (f) Lookup $SK[25, 19]$ in $\mathcal{H}_{1,(1,3),(1,3)}$ to obtain $SK[20, 30]$.
   (g) Match on Byte $SK[30]$, which yields a one-byte filter. Discard the current guess if the values do not match.
   (h) Lookup $SK[24, 18]$ in $\mathcal{H}_{1,(0,2),(0,2)}$ to obtain $SK[23, 29]$.
   (i) Lookup $SK[26, 20]$ in $\mathcal{H}_{1,(1,3),(0,1)}$ to obtain $SK[23, 29]$.
   (j) Match on Bytes $SK[23, 29]$, which yields a two-byte filter. Discard the current guess if the values do not match.
2. For all 32-bit values of $SK[0, 1, 2, 5]$:
   (a) Lookup $SK[0, 5]$ in $\mathcal{H}_{1,(0,1),(0,2)}$ to obtain $SK[10, 15]$.
   (b) Lookup $SK[2, 15]$ in $\mathcal{H}_{2,(0,1),(0,1)}$ to obtain $SK[9, 12]$.
   (c) Lookup $SK[12, 1]$ in $\mathcal{H}_{1,(0,1),(1,3)}$ to obtain $SK[6, 11]$.
   (d) Lookup $SK[6, 0]$ in $\mathcal{H}_{2,(0,1),(0,1)}$ to obtain $SK[3, 13]$.
   (e) Lookup $SK[9, 3]$ in $\mathcal{H}_{1,(1,3),(1,3)}$ to obtain $SK[4, 14]$.
   (f) Lookup $SK[11, 5]$ in $\mathcal{H}_{1,(1,3),(0,1)}$ to obtain $SK[8, 14]$.
   (g) Match on Byte $SK[14]$, which yields a one-byte filter. Discard the current guess if the values do not match.

(h) Lookup $SK[10,1]$ in $\mathcal{H}_{2,(0,3),(0,1)}$ to obtain $SK[4,7]$.
(i) Match on Byte $SK[4]$, which yields a one-byte filter.
Discard the current guess if the values do not match.
(j) Lookup $SK[13,2]$ in $\mathcal{H}_{1,(1,2),(0,2)}$ to obtain $SK[7,8]$.
(k) Match on Bytes $SK[7,8]$, which yields a two-byte filter.
Discard the current guess if the values do not match.

E.2.4.3 Complexity

The attack involves 32 key bytes, i.e., 256 bits. The probability for each ciphertext pair to have $(\Delta C_2, \Delta C_3) = (0,0)$, i. e. 32 fixed inactive ciphertext difference bytes, is approximately 2^{-256}. The probability that, for a given plaintext-ciphertext pair and key candidate, the input and output differences to the impossible differential trails are fulfilled is $36 \cdot 2^{-256}$. Thus, each pair that survives this initial ciphertext-side filter yields $36 \approx 2^{5.2}$ keys on average. To have the probability that a key is not filtered after the attack, $(1-2^{-256})^N$, sufficiently small with $N \approx 2^{263}$, we need approximately $N \approx 1/2^{-256+5.2-7} = 2^{257.8}$ pairs that survive the initial ciphertext-pair filter. This implies that we have $2^{256+257.8} = 2^{513.8}$ pairs in total. Since a structure of 2^{256} texts can yield 2^{511} pairs, this implies $2^{2.8} \approx 7$ structures will be necessary, which corresponds to $2^{258.8}$ CPs.

The complexity consists of:

- $2 \cdot 36 \cdot 2^{32} \cdot 2^{16} \approx 2^{54.2}$ precomputations through a single round of AES;
- Encrypting $2^{258.8}$ CPs;
- Writing them into and reading them from $\mathcal{C}$ with ca. $2 \cdot 2^{258.8} \approx 2^{259.8}$ MAs;
- Extracting $2^{2.8+511-256} = 2^{257.8}$ pairs that survive the first filter;
- Extracting the corresponding keys with $2^{257.8} \cdot 36 \cdot (2^{24} + 2^{32}) \cdot 8 \approx 2^{298.0}$ MAs to the precomputed tables;
- Filtering the keys down to 36 valid keys on average and setting 36 flags in $\mathcal{K}$ with $2^{263.0}$ MAs on average;
- Extracting all surviving key candidates with 2^{256} MAs, and
- Testing the surviving keys with one encryption each.

We expect $2^{256} \cdot (1-2^{-256})^{2^{263}} \approx 2^{71.3}$ candidates to survive. We can repeat the attack with the trails shifted to cover the secret-key bytes $SK[32..63]$. Then, with twice the complexity, one can expect $2^{2 \cdot 71.3} \approx 2^{142.6}$ candidates to remain that can be tested with a single encryption each.

In total, those complexities sum to $2^{54.2} + 2 \cdot 2^{258.8} + 2^{142.6} \approx 2^{259.8}$ EEs plus $2 \cdot (2^{259.8} + 2^{298.0} + 2^{263.0}) \approx 2^{299}$ MAs to tables that we approximate with at least one ten-round encryption each. The attack needs memory for 2^{256} bits of keys or 2^{247} states of key candidates plus 2^{257} plaintext-ciphertext pairs for a structure at a time, i.e., ca. 2^{257} states. In total, the attack needs approximately:

- Data-collection phase: $2^{259.8}$ EEs, $2^{260.8}$ MAs, $2^{259.8}$ CPs.
- Memory: 2^{257} states.
- Attack phase: $2^{299.0}$ MAs to large tables.

E.2.5 Impossible-differential Attack on 10-round Vistrutah-512-256

We can use similar trails as for Vistrutah-512 also for an impossible-differential attack on 10 rounds of Vistrutah-512-256.

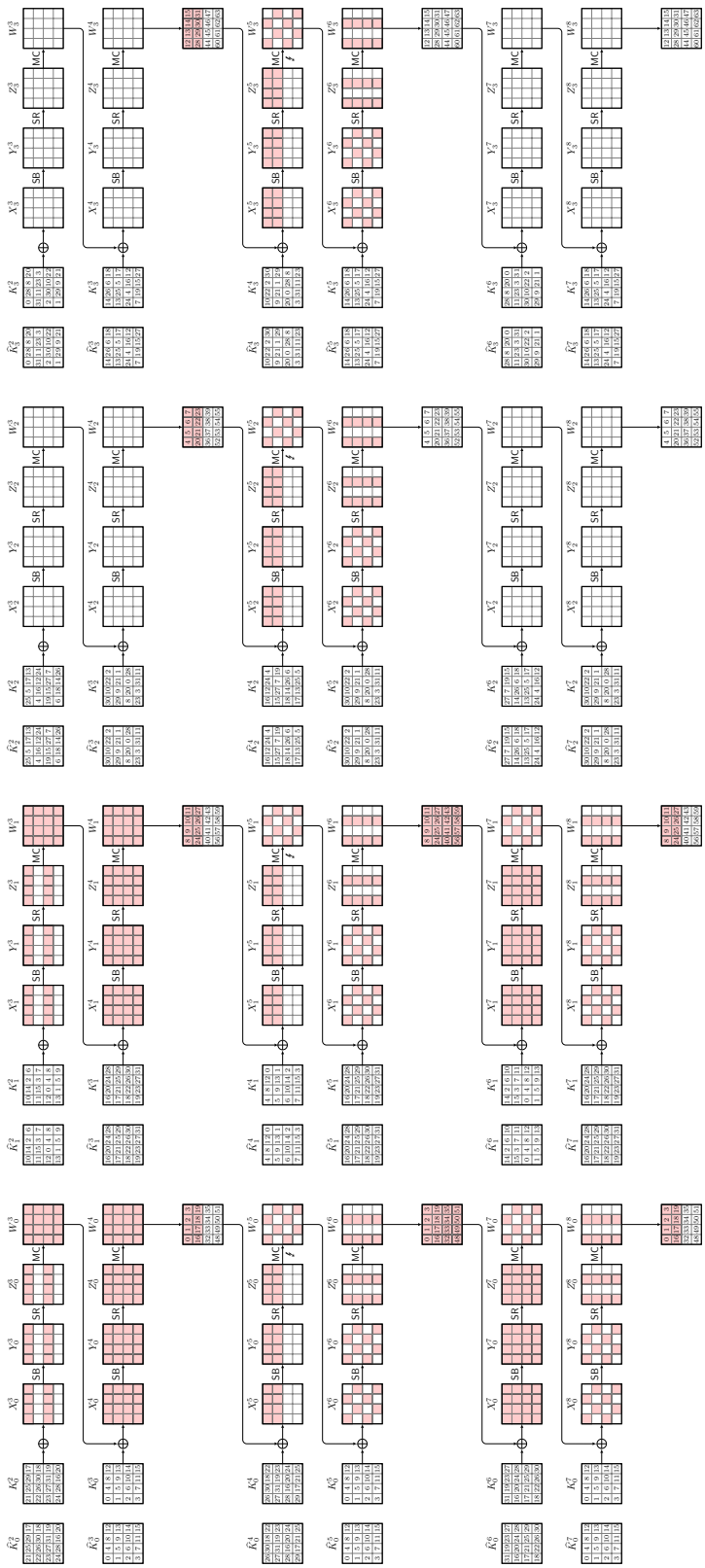


Figure 25: Impossible-differential trail in the attack on 10-round `Vistrutah`-512-256.

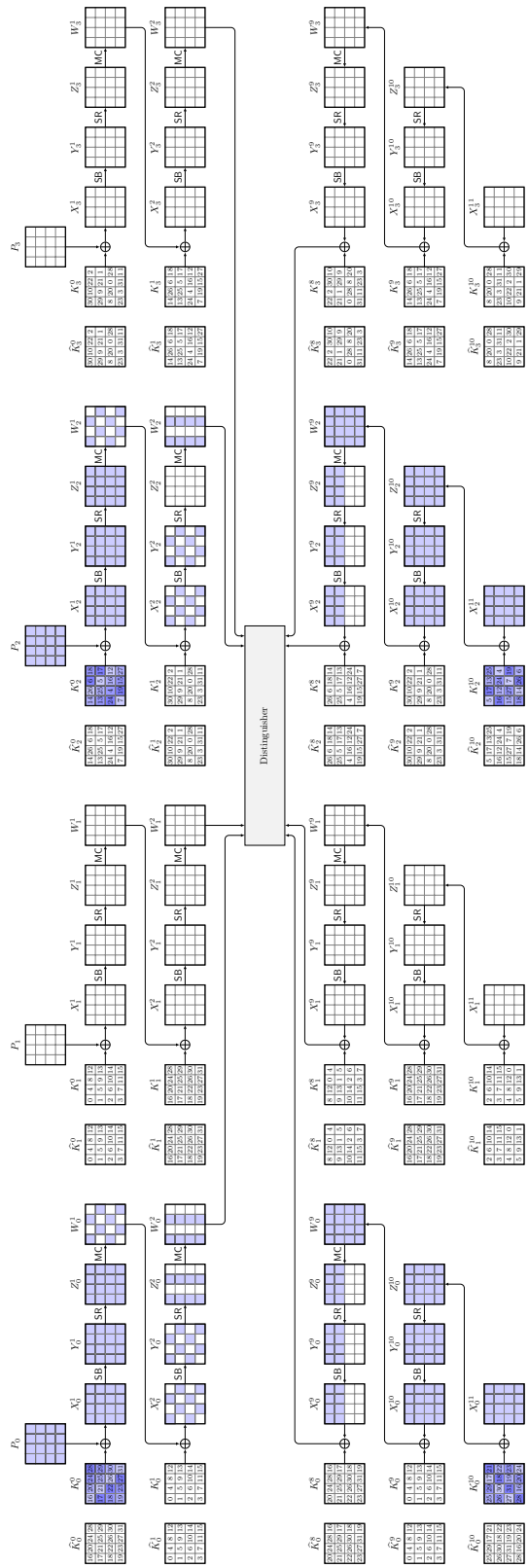


Figure 26: The key-recovery rounds of the 10-round impossible-differential attack on Vistrutah-512-256.

E.2.5.1 Trails

We slightly change the slices used in the key-recovery phase to exploit that the K_2^0 or K_3^0 overlap in eight bytes each with K_0^0 and K_1^0. We can exploit the fact that K_2^{10} shares the same secret-key bytes as K_2^0. The slighly adapted impossible-differential trail is shown in Figure 25, covering Rounds 3 through 8. The key-recovery phase is depicted in Figure 26. Due to the overlap, the attack involves 24 key bytes. $SK[4..7, 12..31]$. We can use at least 36 trails as before:

- Any combination of $\binom{4}{2}$ pairs of active columns of ΔW_0^2 and the same columns in ΔW_2^2 such that exactly two 128-bit slices will be active after the mixing layer in ΔX^3. All such trails lead to at most two active bytes in any column in ΔZ^5.
- Similarly, we can use any combination of $\binom{4}{2}$ pairs of active rows of ΔX_0^9 and the same rows in ΔX_2^9 such that at most two 128-bit slices will be active after the inverse mixing layer in ΔW^8. All such trails lead to at most two active bytes in any column in ΔW^5, which contradicts with ΔZ^5 due to the `MixColumns` branch number.

E.2.5.2 Attack

Again, a few lookup tables must be precomputed:

- 36 tables $\mathcal{H}_{1,(i,j),(i',j')}$ for all $(i,j),(i',j') \in \{0,1,2,3\}^{2\cdot 2}$ with $i \neq j$ and $i' \neq j'$: for a four-byte input difference $(\alpha_0, \alpha_1, \alpha_2, \alpha_3)$, outputs all values that produce a column difference of $(\beta_0, \beta_1, \beta_2, \beta_3)$ after one round of `AES` for any $\alpha_i, \alpha_j, \beta_{i'}, \beta_{j'} \in (GF(2^8))^4$ but all other byte differences being zero.
- 36 tables $\mathcal{H}_{2,(i,j),(i',j')}$ for all $(i,j),(i',j') \in \{0,1,2,3\}^{2\cdot 2}$ with $i \neq j$ and $i' \neq j'$: $(\alpha_0, \alpha_1, \alpha_2, \alpha_3)$, outputs all values that produce a column difference of $(\beta_0, \beta_1, \beta_2, \beta_3)$ after one inverse round of `AES` for any $\alpha_i, \alpha_j, \beta_{i'}, \beta_{j'} \in (GF(2^8))^4$ but all other byte differences being zero.

Then, the attacks consists of the following steps:

1. Initialize a list $\mathcal{K}$ of $2^{24\cdot 8}$ 0-bits for the involved key bytes.
2. Initialize an empty hash table of ciphertexts $\mathcal{C}$.
3. Choose 2^s structures of 2^{256} plaintexts that iterate over all values in the first two 128-bit diagonals of all plaintext slices, with all other bytes assigned to random values that are the same for all plaintexts.
4. For all plaintexts, ask for their corresponding encryptions C and store them into $\mathcal{C}$ indexed by integer representations of their 32-byte values in their ciphertext slices (C_1, C_3).
5. Identify the pairs at equal indices in $\mathcal{C}$.
6. For each such pair and for each of the 36 trail options, derive the keys that produce an impossible trail. We isolate an example description for the trail in Figure 26 in the next list below.
7. For each 24-byte key option that leads to an impossible differential, set its bit to 1 in $\mathcal{K}$.
8. Discard all keys with 1 bits in $\mathcal{K}$ as impossible.
9. For all nonmarked keys, guess the remaining eight key bytes and test each candidate with one encryption each.
10. Output the keys that satisfy this test.

For the sake of brevity, we only describe steps for finding the impossible keys for the trail in Figure 26 in the following, as they are similar for the other trails. The procedure is repeated 6^2 times for all possible trails.

We can partition the keys $K_0^0 = SK[16..31]$ and $K_2^{10} = SK[4..7, 12..19, 24..27]$ into

their four diagonals each and the keys $K_0^{10} = SK[16..31]$ and $K_2^{10} = SK[4..7, 12..19, 24..27]$ into their four inverse diagonals each. From analyzing the combinations of shared secret-key bytes, we obtained combinations as shown in matching shades in those keys in Figure 26.

1. For all 16-bit values of $SK[16, 25]$:
   (a) Lookup $SK[25, 16]$ in $\mathcal{H}_{1,(1,2),(0,2)}$ to obtain $SK[14, 27]$.
   (b) Lookup $SK[25, 27]$ in $\mathcal{H}_{2,(0,2),(0,1)}$ to obtain $SK[18, 24]$.
   (c) Lookup $SK[25, 16]$ in $\mathcal{H}_{2,(0,3),(0,1)}$ to obtain $SK[19, 22]$.
   (d) Lookup $SK[25, 19]$ in $\mathcal{H}_{1,(1,3),(1,3)}$ to obtain $SK[20, 30]$.
   (e) Lookup $SK[27, 24]$ in $\mathcal{H}_{2,(2,3),(0,1)}$ to obtain $SK[17, 30]$.
   (f) Match on Byte $SK[30]$, which yields a one-byte filter. Discard the current guess if the values do not match.
   (g) Lookup $SK[22, 27]$ in $\mathcal{H}_{1,(2,3),(1,3)}$ to obtain $SK[17, 28]$.
   (h) Match on Byte $SK[17]$, which yields a one-byte filter. Discard the current guess if the values do not match.
   (i) Lookup $SK[24, 19]$ in $\mathcal{H}_{1,(2,3),(0,2)}$ to obtain $SK[6, 17]$.
   (j) Match on Byte $SK[17]$, which yields a one-byte filter. Discard the current guess if the values do not match.
   (k) Lookup $SK[16, 19]$ in $\mathcal{H}_{2,(1,2),(0,1)}$ to obtain $SK[17, 26]$.
   (l) Match on Byte $SK[17]$, which yields another one-byte filter. Discard the current guess if the values do not match.
   (m) Lookup $SK[16, 26]$ in $\mathcal{H}_{1,(0,2),(0,2)}$ to obtain $SK[21, 31]$.
   (n) Lookup $SK[18, 28]$ in $\mathcal{H}_{2,(1,3),(0,1)}$ to obtain $SK[21, 31]$.
   (o) Match on Bytes $SK[21, 31]$, which yields a two-byte filter. Discard the current guess if the values do not match.
   (p) Lookup $SK[24, 18]$ in $\mathcal{H}_{1,(0,2),(0,2)}$ to obtain $SK[23, 29]$.
   (q) Lookup $SK[26, 20]$ in $\mathcal{H}_{2,(1,3),(0,1)}$ to obtain $SK[23, 29]$.
   (r) Match on Bytes $SK[23, 29]$, which yields a two-byte filter. Discard the current guess if the values do not match.
   (s) For all 8-bit values of $SK[5]$:
      i. Lookup $SK[26, 5]$ in $\mathcal{H}_{1,(0,1),(1,3)}$ to obtain $SK[7, 12]$.
      ii. Lookup $SK[5, 14]$ in $\mathcal{H}_{2,(0,3),(0,1)}$ to obtain $SK[4, 7]$.
      iii. Match on Byte $SK[7]$, which yields a one-byte filter. Discard the current guess if the values do not match.
      iv. Lookup $SK[18, 4]$ in $\mathcal{H}_{1,(0,2),(1,3)}$ to obtain $SK[13, 15]$.
      v. Lookup $SK[12, 6]$ in $\mathcal{H}_{2,(1,3),(0,1)}$ to obtain $SK[13, 15]$.
      vi. Match on Bytes $SK[13, 15]$, which yields a two-byte filter. Discard the current guess if the values do not match.

E.2.5.3 Complexity

The attack involves 24 key bytes, i.e., 192 bits. We query in total three structures of $3 \cdot 2^{256} \approx 2^{257.6}$ CPs that form ca. $3 \cdot 2^{511}$ pairs, among which $3 \cdot 2^{511-256} \approx 2^{256.6}$ pairs survive the initial filter on ciphertext side to be active in only two state slices.

The first round has $2 \cdot 8$ byte conditions and so has the final round, i.e., 32 byte conditions. The probability for a key candidate to lead to an impossible differential is approximately $6^2 \cdot 2^{-256} = 2^{-250.8}$. From the data, the probability that a specific key is not filtered after the attack is approximately $(1 - 2^{-250.8})^{256.6} \approx 2^{-80.4}$. The complexity consists of:

- $2 \cdot 36 \cdot 2^{32} \cdot 2^{16} \approx 2^{54.2}$ precomputations through a single round of `AES`;
- Encrypting $2^{257.6}$ CPs;
- Writing them into and reading them from $\mathcal{C}$ with $2 \cdot 3 \cdot 2^{256} \approx 2^{258.6}$ MAs;
- Extracting $3 \cdot 2^{511-256} \approx 2^{256.6}$ pairs that survive the first filter;
- Extracting and filtering the corresponding keys with ca. $2^{256.6} \cdot 36 \cdot 2^{16} \cdot (5 + 2^{-8} \cdot 11) \approx 2^{280.1}$ MAs to the precomputed tables;
- Setting $2^{-64} \cdot 36 \cdot 2^{256.6} \approx 2^{197.8}$ flags in $\mathcal{K}$;
- Extracting all surviving key candidates with 2^{192} MAs; and
- Testing the approximately $2^{192} \cdot 2^{-80.4} \approx 2^{111.6}$ surviving key candidates with 2^{64} encryptions each.

Thus, we obtain $2^{54.2} + 2^{257.6} + 2^{175.6} \approx 2^{257.6}$ EEs and $2^{258.6} + 2^{280.1} + 2^{197.8} + 2^{192} \approx 2^{280.1}$ MAs. The attack needs memory for 2^{192} bits of keys or $2^{190.6}$ states of key candidates plus 2^{257} plaintext-ciphertext pairs for a structure at a time, i.e., 2^{257} states. In total, the attack needs approximately

- Data-collection phase: $2^{257.6}$ EEs, $2^{258.6}$ MAs, $2^{257.6}$ CPs.
- Memory: 2^{257} states.
- Attack phase: $2^{280.1}$ MAs to large tables.

E.2.6 Impossible-differential Attack on 14-round `Vistrutah`-512

We adapt Mala et al.'s impossible-differential attack on seven-round `AES`-128 [MDRM10], used in variants with single or multiple trails and revised analyses for the `AES` [BNS14, BLNS18, LP21, LP25]. We obtain an impossible-differential attack on 14 rounds of `Vistrutah`-512 without the final `MixColumns` and starting from the beginning of Round 2.

E.2.6.1 Distinguisher

The core of the adapted distinguisher is depicted in Figure 27. Its final round and the key-recovery phase are shown in Figure 28. The trail is a direct adaptation of the one from [MDRM10] to `Vistrutah`'s larger state. It starts from two states with the same three active diagonals at the beginning of Round 6 that is mapped through 3.5 rounds to each slice having at most three active bytes in each column at ΔZ^9. At the bottom, the distinguisher starts with a single state slice and a single active row in ΔZ_i^{13} that is mapped in decryption direction to ΔW_i^9 having at most one active byte in each column. Due to the branch number of the `MixColumns` matrix, the transition from ΔZ^9 to ΔW^9 is impossible.

E.2.6.2 Key-recovery trails

The attack could use multiple trails, but for the sake of simplicity, we describe an attack with one trail and therefore a single in- and outbound phase. We forms structures of 2^{256} CPs that iterate over all values in the first and third diagonals of all state slices and leaves all other plaintext diagonals random but constant over all plaintexts of a structure. Such a structure can produce approximately 2^{511} ciphertext pairs. Since the probability for a ciphertext pair to survive the initial filter at ΔZ^{15} is approximately 2^{-384}, we can expect about $2^{511-384} = 2^{127}$ surviving pairs per structure.

At the plaintext side, we want ΔW_i^3 in the two active slices has only two diagonals active. The active diagonals must share the same positions for both active slices. Then, there are only two active slices in ΔX^5 with only two active rows. The attack here requires that both transitions ΔZ_i^5 to ΔW_i^5 result in the latter having one inactive diagonal, which

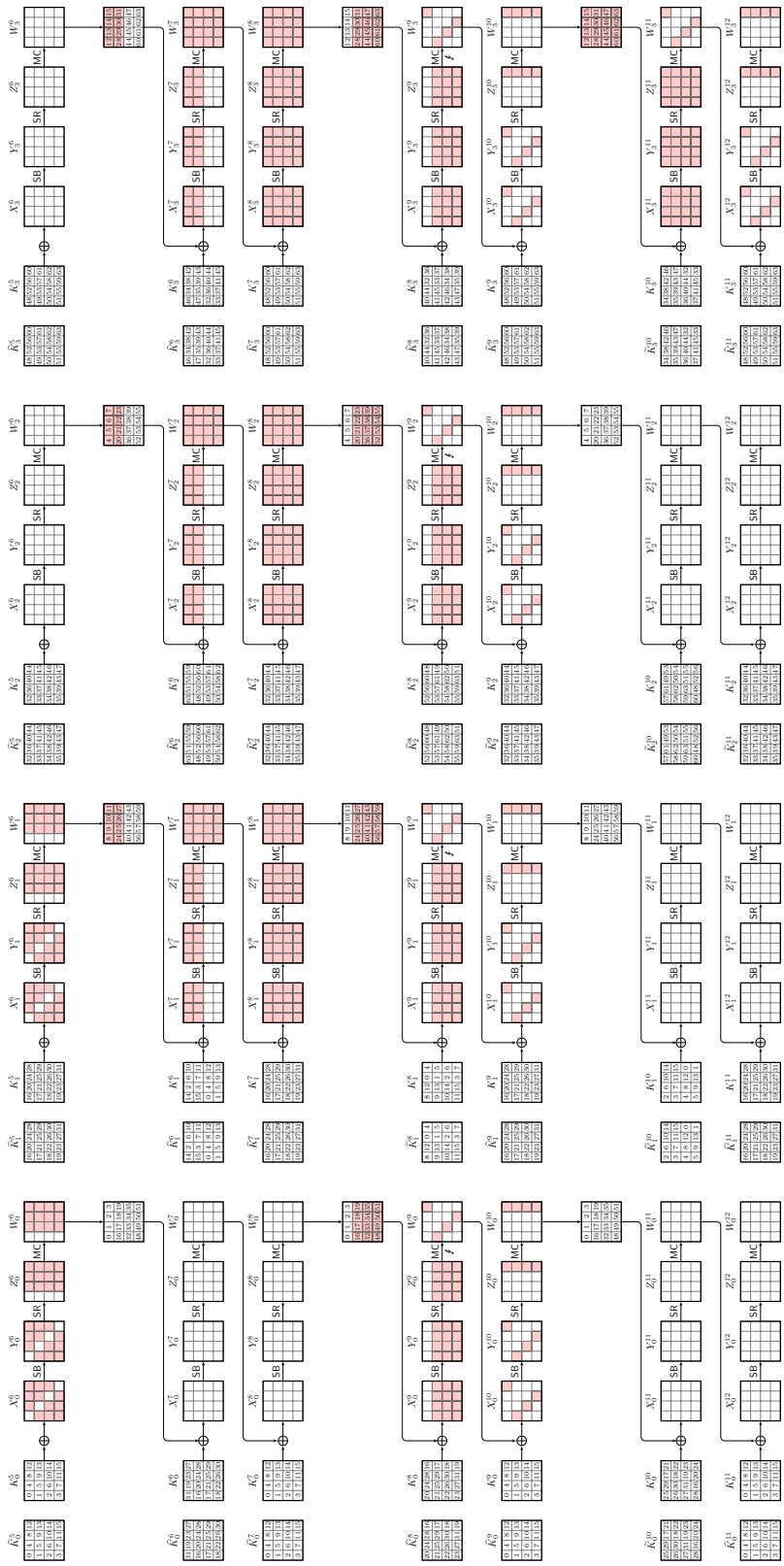


Figure 27: The first seven of the eight-round distinguisher in the impossible-differential attack on 14-round Vistrutah-512.

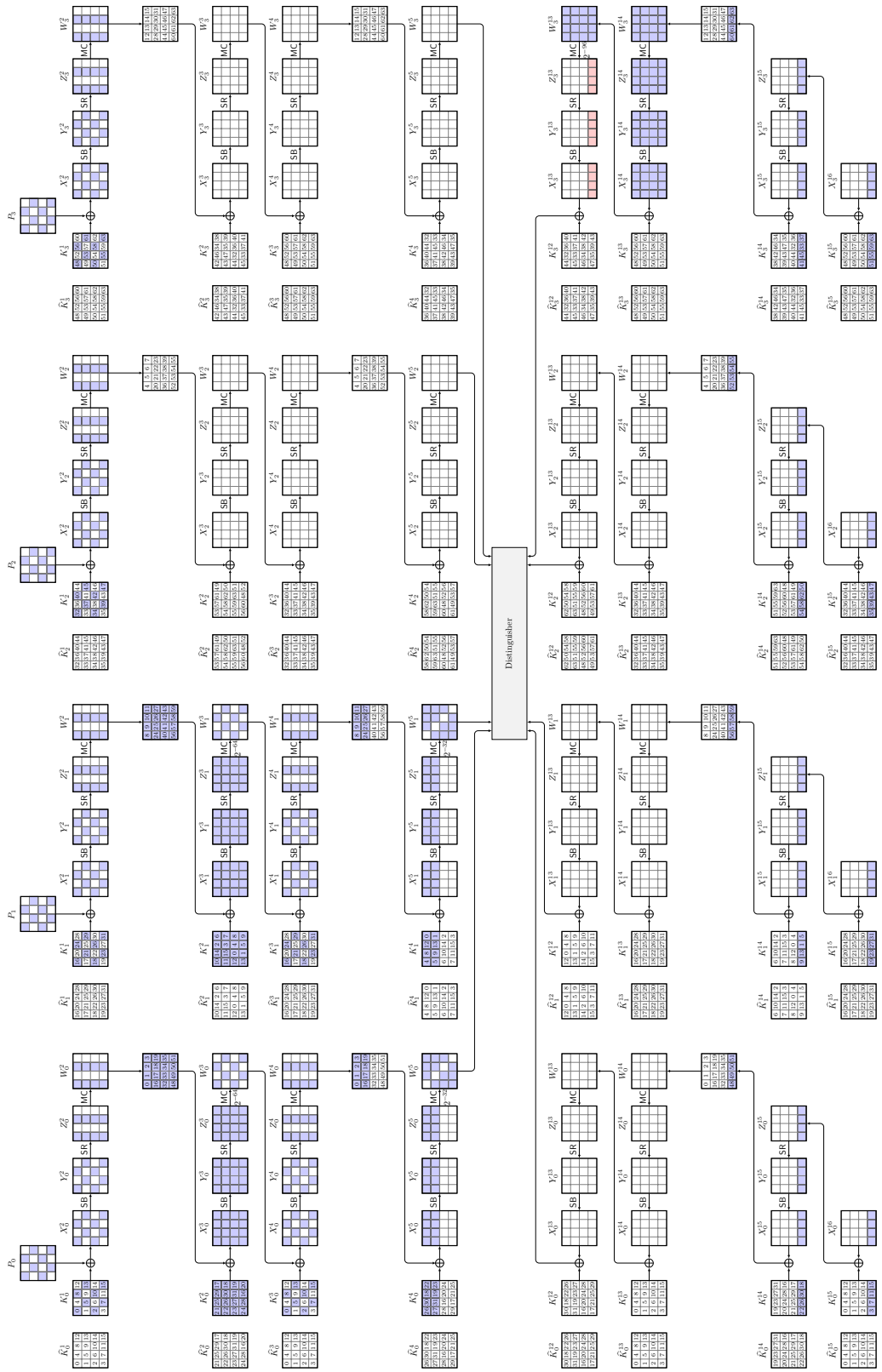


Figure 28: The outer rounds of the online filtering and key-recovery phase for the 14-round impossible-differential attack on `Vistrutah`-512.

should be the same diagonal position for both active slices. Fixing a specific truncated-differential trail, the probability for a key to follow to the impossible input difference is about $2^{-64-64-32-32} = 2^{-192}$, i.e., there are ca. $c_{\text{in}} = 192$ bit conditions.

At the ciphertext side, we search for pairs that possess only one and the same active row in all state slices ΔZ_i^{15}. Moreover, it needs that after partial decryption to the single active slice ΔZ_i^{13}, it has only a single active row. Then, the resulting trail through the middle is impossible and the keys that have suggested those differences can be discarded. The probability for a pair and key to lead to the impossible output difference in decryption direction is approximately 2^{-96}, i.e., there are ca. $c_{\text{out}} = 96$ bit conditions. Thus, there are $c_{\text{in}} + c_{\text{out}} = 192 + 96 = 288$ bit conditions in the trail.

The key-recovery part from the plaintexts to ΔW^5 forms a set of key bytes $\mathcal{K}_{in}$ that involves a total of 48 key bytes as can be seen from Figure 28: $SK[0..15]$, $SK[16..31]$, $SK[32, 34, 37, 39, 40, 42, 45, 47]$, and $SK[48, 50, 53, 55, 56, 58, 61, 63]$. The key-recovery part from the ciphertexts to ΔZ^{13} forms a set of key bytes $\mathcal{K}_{\text{out}}$ that involves a total of 32 key bytes given from the bottom rows of all key slices K_i^{14} and K_i^{15} for all $i \in \{0, 1, 2, 3\}$. The union set of secret-key bytes in both plain- and ciphertext side $\mathcal{K} = \mathcal{K}_{\text{in}} \cup \mathcal{K}_{\text{out}}$ contains a set of in total 56 bytes, i.e., 448 key bits, composed of all secret key bytes *except for* $SK[36, 38, 44, 46, 49, 52, 57, 60]$.

The probability for a key to be suggested by a surviving pair is $2^{-(c_{\text{in}}+c_{\text{out}})} \approx 2^{-288}$. At the end of the attack, we aims to have as few key candidates as possible to survive. For $N = 2^{295}$, we obtain $(1 - 2^{-288})^N \approx 2^{-184.7}$. Thus, we can expect on average $2^{448-184.7} = 2^{263.3}$ key candidates to survive that can be tested with 2^{64} encryptions for the remaining eight bytes each, which yields $2^{327.3}$ encryptions on average. Such a value of 2^{295} surviving pairs requires $2^{295-127} = 2^{168}$ structures of 2^{256} texts each, which implies a data complexity of 2^{424} CPs.

E.2.6.3 Attack

First, several huge tables must be precomputed in the course of the attack. For $\mathcal{H}_1$, iterate over all $2^{36\cdot8}$ options of $\Delta Z_3^{13}[3, 7, 11, 15]$, ΔY_3^{14}, and $K_i^{14}[3, 7, 11, 15]$ for all $i \in \{0, 1, 2, 3\}$ to conduct the following steps:

- Compute from X_3^{14} to $Z_i^{15}[3, 7, 11, 15]$ and $\Delta X_i^{16}[3, 7, 11, 15]$ for all $i \in \{0, 1, 2, 3\}$.
- For the four bytes in K^{14} corresponding to $SK[7, 23, 39, 55]$, compute to $X_i^{16}[7]$.
- Store the key information into $\mathcal{H}_1$ indexed by $\Delta X_i^{16}[3, 7, 11, 15]$.

$\mathcal{H}_1$ contains $2^{36\cdot8}$ entries in total, each of which is stored under a 16-byte index so that we can expect $2^{20\cdot8}$ entries per index on average. The computation of $\mathcal{H}_1$ requires $4 \cdot 2 \cdot 2^{36\cdot8} \cdot (16 + 16)/(14 \cdot 64) \approx 2^{286.2}$ EEs, $2^{36\cdot8}$ MAs, and memory for 2^{288} entries.

The construction of $\mathcal{H}_2$ through $\mathcal{H}_5$ will be conducted for each surviving plaintext pair on-the-fly when those pairs are known in the attack. We only provide the description for their construction here for better overview. In the following, we assume a fixed pair of first diagonals in all plaintext slices is given.

For $\mathcal{H}_2$ and for all $2^{24\cdot8}$ options for the 16 bytes in the first diagonals of K_i^1 the eight active bytes in ΔW_0^3:

- Compute from the plaintexts to the full W_0^3. Since four bytes of K_0^2 overlap with K_1^1, filter out invalid candidates to obtain $2^{20\cdot8}$ candidates on average. Note that this can be conducted efficiently with a sub-meet-in-the-middle procedure in $O(2^{20\cdot8})$ computations.
- Guess $K_0^3[2, 7, 8, 13] = SK[2, 7, 8, 13]$. Note that at this point we have $K_0^3[0, 5, 10, 15] = = K_0^1[0, 5, 10, 15]$ and the full K_0^2. From the known active bytes in W_0^3 and its difference, compute to $Z_0^5[0, 4, 8, 12]$ and $\Delta Z_0^5[0, 4, 8, 12]$, derive the full ΔW_0^5 and the full ΔY_0^5 backwards.

- Guess $K_1^4[0,8] = SK[4,12]$. From $W_0^4[8..11]$ and $\Delta W_0^4[8..11]$, $K_1^4[0,8]$ and $K_1^4[4,12] = SK[0,8] = K_0^3[8,0]$, we can compute $Z_1^5[0,4,8,12]$ and $\Delta Z_1^5[0,4,8,12]$, derive the full ΔW_0^5 and the full ΔY_1^5 backwards.
- Store the keys, $Y_0^5[0,4,8,12]$, and $Y_1^5[0,4,8,12]$, indexed by the 20 bytes of $SK[0,\ 2,\ 4,\ 5,\ 7,\ 8,\ 10,\ 12,\ 13,\ 15,\ 16,\ 18,\ 19,\ 21,\ 23,\ 24,\ 26,\ 27,\ 29,\ 31]$, $\Delta Y_0^5[0,\ 1,\ 4,\ 5,\ 8,\ 9,\ 12,\ 13]$, and $\Delta Y_1^5[0,\ 1,\ 4,\ 5,\ 8,\ 9,\ 12,\ 13]$. Note that beyond those, it stores $SK[16..31,\ 32,\ 37,\ 42,\ 47,\ 48,\ 53,\ 58,\ 63]$.

Computing $\mathcal{H}_2$ requires $2^{20\cdot 8} \cdot 2 \cdot (16+16)/896 \approx 2^{156.2}$ EEs for Rounds 2 and 3 plus $2^{26\cdot 8} \cdot 2 \cdot (8+8+8)/896 \approx 2^{203.8}$ EEs, $2^{26\cdot 8}$ MAs, and memory for $2^{26\cdot 8}$ entries on average.

For $\mathcal{H}_3$ and for all $2^{24\cdot 8}$ options for the 16 bytes in the third diagonals of K_i^1 and the eight active bytes in ΔW_1^3:

- Compute from the plaintexts to the full W_1^3. Since four bytes of K_1^2 overlap with K_0^1, filter out invalid candidates to obtain $2^{20\cdot 8}$ candidates on average. Again, this can be conducted efficiently with a sub-meet-in-the-middle procedure in $O(2^{20\cdot 8})$ computations.
- Guess $K_1^3[0,5,10,15] = SK[16,21,26,31]$. From the known active bytes in W_1^3 and its difference, compute to $Z_1^5[1,5,9,13]$ and $\Delta Z_1^5[1,5,9,13]$, derive the full ΔW_1^5 and the full ΔY_1^5 backwards.
- Guess $K_0^4[1,5] = SK[27,19]$. From $W_1^4[0..3]$ and $\Delta W_1^4[0..3]$, and $K_0^4[1,5,9,13]$, we can compute $Z_0^5[1,5,9,13]$ and $\Delta Z_0^5[1,5,9,13]$, derive the full ΔW_0^5 and the full ΔY_0^5 backwards.
- Store the keys, $Y_0^5[1,5,9,13]$, and $Y_1^5[1,5,9,13]$, indexed by the 20 bytes of $SK[0,\ 2,\ 4,\ 6,\ 7,\ 8,\ 10,\ 12,\ 13,\ 15,\ 16,\ 18,\ 19,\ 21,\ 23,\ 24,\ 26,\ 27,\ 29,\ 31]$, $\Delta Y_0^5[0,\ 1,\ 4,\ 5,\ 8,\ 9,\ 12,\ 13]$, and $\Delta Y_1^5[0,\ 1,\ 4,\ 5,\ 8,\ 9,\ 12,\ 13]$. Note that beyond those, it stores $SK[0..15,\ 34,\ 39,\ 40,\ 45,\ 50,\ 55,\ 56,\ 61]$.

Computing $\mathcal{H}_3$ requires $2^{20\cdot 8} \cdot 2 \cdot (16+16)/896 \approx 2^{156.2}$ EEs for Rounds 2 and 3 plus $2^{26\cdot 8} \cdot 2 \cdot (8+8+8)/896 \approx 2^{203.8}$ EEs, $2^{26\cdot 8}$ MAs, and memory for $2^{26\cdot 8}$ entries on average.

Merge both tables $\mathcal{H}_2$ and $\mathcal{H}_3$ into a table $\mathcal{H}_4$, indexed by the 24 bytes of $SK[1,\ 3,\ 5,\ 7,\ 9,\ 11,\ 13,\ 15,\ 18,\ 19,\ 22,\ 23,\ 26,\ 27,\ 30,\ 31,\ 37,\ 39,\ 45,\ 47,\ 50,\ 55,\ 58,\ 63]$. In each entry we store the known 48 bytes from $\mathcal{K}_{\text{in}}$ of the secret key. Given $2^{26\cdot 8}$ entries in $\mathcal{H}_2$ and $2^{26\cdot 8}$ entries in $\mathcal{H}_3$ and a 28-byte index that is equal for both tables, we expect $2^{24\cdot 8}$ entries to match. We approximate the merging effort to require $2^{26\cdot 8} \cdot 3 \approx 2^{209.6}$ MAs, and memory for $2^{24\cdot 8}$ merged entries.

Given a ciphertext pair, retrieve the $2^{20\cdot 8}$ entries from $\mathcal{H}_1$. Each entry defines a 32-byte value for the key bytes of $\mathcal{K}_{\text{out}}$ that include the 24 bytes used as the index of $\mathcal{H}_4$. We expect this step to require $2^{20\cdot 8} \cdot 3 \approx 2^{129.6}$ MAs, and memory for $2^{20\cdot 8}$ entries.

Compute a table $\mathcal{H}_5$ from merging $\mathcal{H}_4$ with the retrieved entries from $\mathcal{H}_1$. Since we merge $2^{20\cdot 8}$ entries from $\mathcal{H}_1$ with $2^{24\cdot 8}$ entries from $\mathcal{H}_4$ and filter for a 24-byte index, we expect $2^{20\cdot 8}$ matching entries on average. We expect this step to require $2^{20\cdot 8} \cdot 3 \approx 2^{161.6}$ MAs, and memory for $2^{20\cdot 8}$ entries.

The final attack proceeds as follows:

1. Initialize a structure $\mathcal{K}$ with one bit set to 0 for each of the values defined by the 56 involved key bytes.
2. Form 2^{168} structures of plaintexts that iterate over all values in the first two diagonals of all state slices and set the remaining bytes in all plaintexts of a structure to a unique random constant.
3. Ask for their corresponding encryptions and store them into a table $\mathcal{C}$.
4. Identify ciphertext pairs differing in only the bottommost rows in the state slices ΔZ_i^{15}.
5. For all so-identified pairs of plaintexts (P, P') and corresponding ciphertexts (C, C'):

- Construct $\mathcal{H}_4$ for the current plaintext pair as described above.
- Lookup from $\mathcal{H}_1$ the $2^{20.8}$ entries for the current ciphertext pair.
- Merge both tables into $\mathcal{H}_5$ that contains 2^{160} impossible key candidates on average.
- Set the flags for each of them to 1 in $\mathcal{K}$.

6. Identify the keys that are not deemed impossible in $\mathcal{K}$. For each of them, guess the remaining 10 key bytes and test each of them with one encryption. Output the candidates that survive this test.

E.2.6.4 Complexity

The attack requires on average

- For $\mathcal{K}$: 2^{448} MAs.
- For $\mathcal{H}_1$: $2^{286.2}$ EEs, 2^{288} MAs, and memory for 2^{288} entries.
- For encryption: $2^{168} \cdot 2^{256} = 2^{424}$ full encryptions, 2^{425} MAs, and memory for 2^{257} states for a structure.
- For each of the 2^{295} surviving pairs (their memory can be reused):
  - For $\mathcal{H}_2$: $2^{203.8}$ EEs, 2^{208} MAs, and memory for 2^{208} entries.
  - For $\mathcal{H}_3$: $2^{203.8}$ EEs, 2^{208} MAs, and memory for 2^{208} entries.
  - For $\mathcal{H}_4$: $2^{209.6}$ MAs and memory for 2^{192} merged entries.
  - For $\mathcal{H}_5$: $2^{161.6}$ MAs, and memory for 2^{160} merged entries.
  - Into $\mathcal{K}$: 2^{160} MAs.
- 2^{448} MAs into $\mathcal{K}$ to find surviving keys.
- $2^{327.3}$ final encryptions.

This yields $2^{286.2} + 2^{295} \cdot \left(2^{203.8} + 2^{203.8}\right) + 2^{327.3} \approx 2^{499.8}$ EEs and $2^{448} + 2^{288} + 2^{295} \cdot \left(2^{208} + 2^{208} + 2^{209.6} + 2^{161.6} + 2^{160}\right) + 2^{448} \approx 2^{505.3}$ MAs plus $2^{448} \cdot 1/512 + 2^{288} + 2^{257} + 2^{208} + 2^{208} + 2^{192} + 2^{160} \approx 2^{439}$ states of memory. In total, the attack needs approximately

- Data-collection phase: 2^{424} EEs, 2^{425} MAs, 2^{424} CPs, and 2^{257} states of memory.
- Online phase: $2^{499.8}$ EEs, $2^{505.3}$ MAs, and 2^{439} states of memory.

E.3 Boomerang Cryptanalysis

E.3.1 Boomerang Attack on 8-round Vistrutah-256

In the following, we describe a boomerang attack on eight rounds of Vistrutah-256.

E.3.1.1 Trails

Figures 29 and 30 show the encryption and decryption truncated-differential trails, respectively, from CPs (P, P') to the ciphertexts (C, C') in blue in Figures 29, and in decryption direction from $(\overline{C}, \overline{C}') = (C \oplus \Delta C, C' \oplus \Delta C)$ back to the middle in red in Figure 30 and to the corresponding plaintexts $(\overline{P}, \overline{P}')$ in blue in Figure 30.

The trail from $P \oplus P'$ to $C \oplus C'$ has a probability of

- ca. $2 \cdot 2^{-64} = 2^{-63}$ to have only bytes at even or only odd indices active at ΔW_0^2,
- similarly, approximately 2^{-63} at ΔW_0^4, and
- similarly, approximately 2^{-63} at ΔW_0^6.

Then, a pair will have only the first two or only the last two anti-diagonals active in each slice of ΔZ^9. Thus, we can invert the final MixColumns operation, add all 2^{128} values to

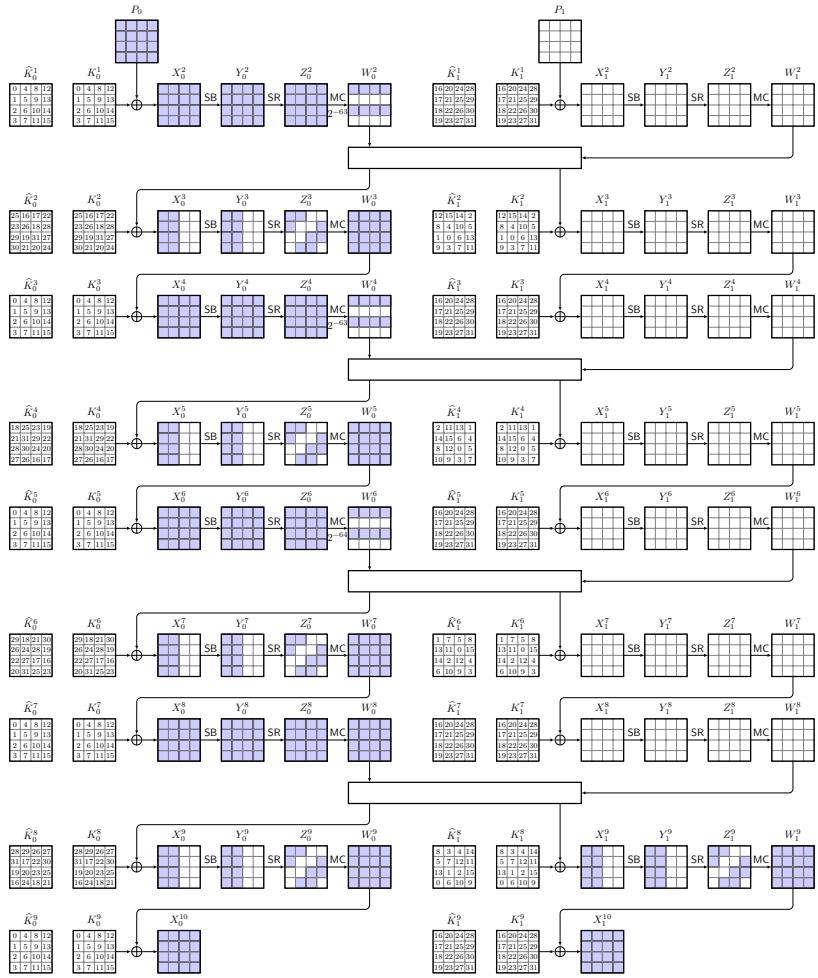


Figure 29: Boomerang attack on eight-round `Vistrutah`-256: Forward trail.

one of the two inactive anti-diagonal in both state slices, and apply `MixColumns` to derive $\overline{C}$ and $\overline{C}'$ from C and C', respectively.

For simplicity, we choose the rightmost two anti-diagonals for that in our description, as shown in Figure 30. The backwards trail then has only the slice ΔW_i^8 active that is inactive in the forward trail. Adding the same value in red bytes to C for deriving $\overline{C}$ as to C' for deriving $\overline{C}'$ means that both pairs have the same values for the red part. If this decryption trail produces only those two columns to be active in ΔX_i^7 (given that only $X_{i\oplus 1}^7$ was active in the forward trail), which occurs with probability approximately 2^{-64}, such that only ΔW_j^6 will be active in the decryption trail that was inactive in the forward trail, both pairs will have the same decryption difference at ΔW_i^4. By locating the boomerang middle phase there, the pair $\overline{W}^4 \oplus \overline{W'}^4$ will also be active only in the same state slice where $W^4 \oplus W'^4$ was active. Thus, it will also be active only in those rows of $\Delta\overline{W}^2$ where ΔW^2 was active. The event in ΔW^2 has to be identified in a key-recovery phase of the initial round key, here K^1 as the attack starts from the beginning of the second round.

The probability of the encryption differential trail is approximately $(2 \cdot 2^{-64})^3 = 2^{-189}$. Since the probability for such a boomerang pair is about $2 \cdot 2^{-128} = 2^{-127}$, we will need multiple pairs for a distinguisher. For this purpose, it will use multiple mixed ciphertext

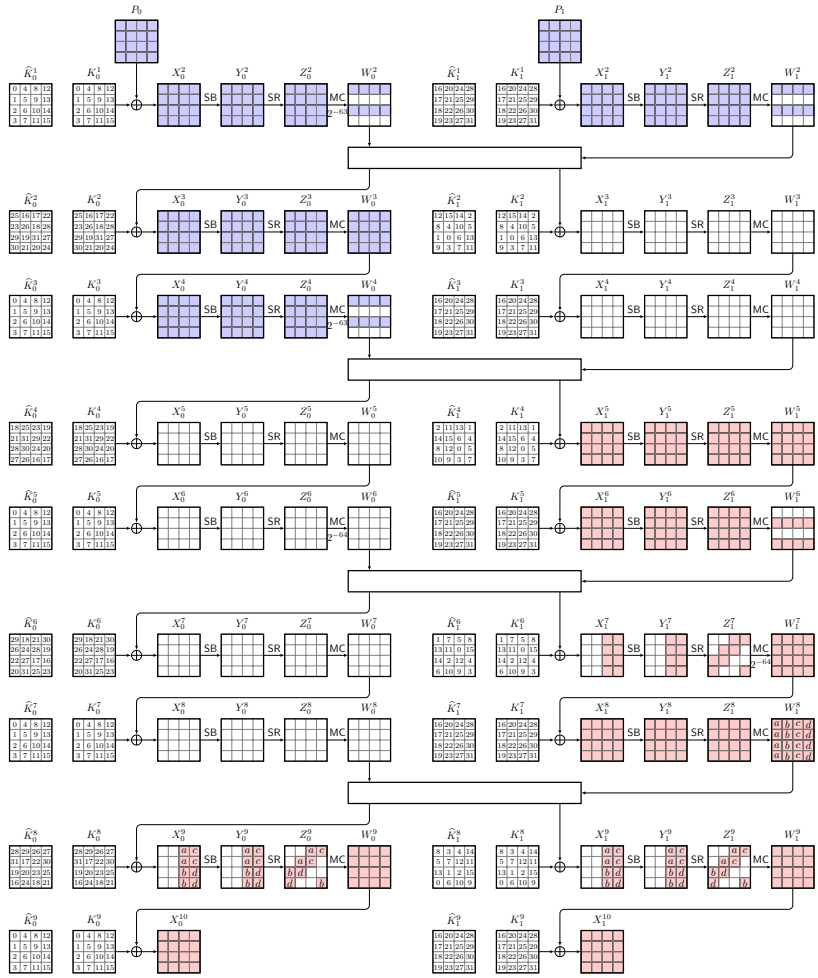


Figure 30: Boomerang attack on eight-round Vistrutah-256: Backward trail.

pairs. Observe that we can partition the eight bytes in the decryption trail at $Z_i^9 \oplus \overline{Z}_i^9$ into four parts of two bytes per slice such that each byte is uniquely mapped to one column in the active slice of W^8. Since we want all mixture pairs $Z_i^7 \oplus \overline{Z}_i^7$ for the active state slice to also have the same two anti-diagonals active, we can produce up to eight mixture pairs by exchanging the slices labeled with a collectively, those with b collectively, and with c and d similarly, as one would do with the four bytes of a plaintext column in the original four-round mixture distinguisher by Grassi [Gra18]. Then, all eight mixture pairs will have the same difference in ΔZ^7, i.e., if one pair follows the desired transition to have only two active anti-diagonals in the active state slice of ΔZ^7 with probability 2^{-64}, then, the seven other pairs are guaranteed to also possess this property with conditional probability one. Since the rest of the truncated-differential decryption trail is deterministic, we obtain up to eight pairs simultaneously that are active each all in only the same two rows of both state slices of $\overline{W}^2 \oplus \overline{W'}^2$.

While the attack targets the full initial key, and therefore the full secret key, we can process each plaintext diagonal separately with eight pairs to reduce the number of key candidates diagonal by diagonal.

E.3.1.2 Attack

The attack steps are as follows.

1. Initialize two precomputation tables $\mathcal{H}_1$ and $\mathcal{H}_2$. For a given four-byte diagonal input difference, $\mathcal{H}_1$ returns all values (2^{16} on average) after the key addition such that they lead to a difference with only Bytes 0 and 2 active after one round. $\mathcal{H}_2$ considers the analogous setting with only Bytes 1 and 3 being active after one round.
2. Initialize two empty tables $\mathcal{C}_0$ and $\mathcal{C}_1$ and eight tables $\mathcal{K}_j$ for $\{0,\ldots,7\}$ for 2^{32} key counters each.
3. Prepare 2^s random but pairwise distinct plaintexts.
4. Ask for the corresponding ciphertexts, invert the final `MixColumns` operation in all slices, and store the resulting plaintext-ciphertext pairs into tables $\mathcal{C}_0$ and $\mathcal{C}_1$ according to their values in the leftmost two anti-diagonals in both state slices (for $\mathcal{C}_0$) or the rightmost two anti-diagonals in both state slices (for $\mathcal{C}_1$).
5. For all ciphertext pairs (C, C') from the same plaintext structure in the same table cell of $\mathcal{C}_0$ (or $\mathcal{C}_1$):
   (a) With their corresponding plaintexts P and P' and $\mathcal{H}_1$ and $\mathcal{H}_2$ derive the 2^{16} potential keys for the first diagonal.
   (b) Choose four sets of c random but pairwise distinct 32-bit values each (e.g. the first such 32-bit value is represented by the label a on four bytes in Z_0^9 and Z_1^9 in Figure 30, the next 32-bit value by the label b and so on) and add each combination to those bytes of Z_0^9 and Z_1^9 that have been inactive in their difference before to derive s^4 pairs $\overline{\widehat{W}}^9$ from $\widehat{W}^9$ and $\overline{\widehat{W'}}^9$ from $\widehat{W'}^9$.
   (c) For all so-derived pairs $\overline{C}$ and $\overline{C}'$, ask for their corresponding decryption to $\overline{P}$ and $\overline{P}'$ and cluster them into $c^4 \cdot (c-1)^4/16$ mixture sets of eight pairs each.
   (d) For all mixture clusters, consider the eight pairs $\overline{P}$ and $\overline{P}'$, use $\mathcal{H}_1$ to derive the potential keys for two mixture pairs for the first diagonal. If the intersection of their surviving key candidates and that for the first diagonal of (P, P') is empty, discard the mixture cluster for $\mathcal{H}_1$. Otherwise, continue the process pair by pair and diagonal by diagonal.
   (e) If no key survives, repeat the process with $\mathcal{H}_2$.
   (f) If a key candidate survives the tests for some (P, P') and all mixture pairs and all diagonals, store the key candidate.
6. For all surviving key candidates, test them with a full encryption and output the keys that satisfy this test as well.

E.3.1.3 Complexities

With $s = 95.5$, i.e., $2^{95.5}$ CPs, we can form 2^{190} pairs. We have a first filter of four inactive anti-diagonals in ΔZ^9 of ca. $2 \cdot 2^{-128} = 2^{-127}$ that produces 2^{63} surviving ciphertext pairs on average. Since the forward trail has probability approximately $8 \cdot 2^{-192} = 2^{-189}$, we can expect 2 correct pairs among them on average. For $c = 400$, we can construct $400^4 \approx 2^{34.6}$ shifted ciphertexts $\overline{C}$ that can be combined to $(400^4 \cdot 399^4)/(2 \cdot 8) = 2^{65.1}$ mixture structures of eight pairs each. Since the event that $\Delta\overline{Z}^7$ in the decryption trail has at most two active anti-diagonals in those anti-diagonals that had been inactive in the forward trail has probability approximately 2^{-64}, we can expect ca. 2 mixture structures that satisfy this property on average for each pair. For those, we can find the secret key with probability one.

For each of the $2^{63} \cdot 2^{65.1} \approx 2^{128.1}$ mixture structures, a wrong key diagonal has probabiliy at most $2 \cdot 2^{32} \cdot (2^{-16})^9$ to survive for the plaintext pair and all eight mixture

pairs for two options of odd or even active rows, i.e., we expect only the correct key to survive.

The complexity consists of

- $2^{95.5}$ encryptions of CPs, the same amount of `MixColumns` inversions, and $2{\cdot}2{\cdot}2^{95.5} \approx 2^{97.5}$ MAs to insert and retrieve pairs into two tables,
- $2^{63} \cdot 8 = 2^{66}$ MAs for all diagonals of the plaintext pair,
- $2^{63} \cdot 2^{34.6} \approx 2^{97.6}$ `MixColumns` operations and 10-round decryptions,
- $2^{63} \cdot 2^{65.1} \cdot 2 \cdot 2 \approx 2^{130.1}$ accesses for the first diagonal and the first two mixture pairs,
- $2^{63} \cdot 2^{65.1} \cdot 2^{-16} \cdot 2 \cdot (6 + 7 \cdot 8) \approx 2^{119.1}$ MAs for the remaing pairs that survived the first test of the first two pairs (2^{-16} on average) into the remaining seven diagonals and for the six remaining pairs for the first diagonal, and
- a negligible amount of full encryptions to test the surviving key candidates.

Thus, the time complexity consists of approximately $2^{95.5}+2^{95.5}\cdot 1/10+2^{97.6}\cdot 1/10+2^{97.6} \approx 2^{98.4}$ EEs and $2^{97.5} + 2^{66} + 2^{130.1} + 2^{119.1} \approx 2^{130.1}$ MAs to tables that we approximate with at least one encryption equivalent. The attack needs memory for $2 \cdot 2 \cdot 2^{95.5} + 2 \cdot 2^{34.6} + 2 \cdot 2^{32} \cdot 2^{16} \cdot 4/32 \approx 2^{97.5}$ states for the ciphertexts (and their plaintexts) in two tables, the decrypted plaintexts in two tables and the two precomputed tables. In total, the attack needs approximately

- Data-collection phase: $2^{95.5}$ CPs and $2^{97.6}$ ACCs.
- Time: $2^{98.4}$ ten-round EEs and $2^{130.1}$ MAs.
- Memory: $2^{97.5}$ states.

E.3.2 Boomerang Attack on 12-round Vistrutah-512

We can mount a variant of the six-round boomerang attack on `AES` by Bariant et al. [BDK+24] (a further development of the result from [BL23]), to obtain an attack on 12 rounds of `Vistrutah`-512, covering Rounds 2 through 13. The attack on six-round `AES` is shown in Figure 31.

E.3.2.1 Original Attack on the AES

The attack employs the shifted retracing boomerang on six rounds from [DKRS20, DKRS24] that was further advanced in [BDK+24]. The forward trail starts from a structure of plaintexts with a single active diagonal and wants several events up to Round 4: one byte inactive in $W^1 \oplus {W'}^1$, an inactive diagonal in $W^3 \oplus {W'}^3$ and two inactive diagonals in $W^4 \oplus {W'}^4$. In the attack, we search for ciphertext pairs C, C' that are active in only two anti-diagonals. For those pairs, we create a ciphertext structure by adding differences to one of the inactive anti-diagonals so that the difference between any $(C, \overline{C})$ is guaranteed to be equal for both pairs automatically, and hopes for having the difference reduced to a single active byte in $Z^5 \oplus \overline{Z}^5$ such that it leads to being active only in the diagonal in $W^3 \oplus \overline{W}^3$ that is inactive in $W^3 \oplus {W'}^3$ and the same holds for the other pairs. Then, having an inactive diagonal in $\overline{W}^1 \oplus \overline{W'}^1$ is guaranteed. As a filter that yielded a beneficial trade-off, the pair further should have a single byte inactive in $\overline{P} \oplus \overline{P}'$. If this property holds for one pair, by creating mixture ciphertext pairs, we can derive several further ciphertext pairs with the same property and derive the initial round key diagonal by diagonal by testing for the inactive byte after the first round.

E.3.2.2 Adapted Attack

We can adapt the attack to `Vistrutah`-512 by scaling up from 8-bit cells to 32-bit cells. However, the diffusion inside the steps requires us to remove the last round, but we can

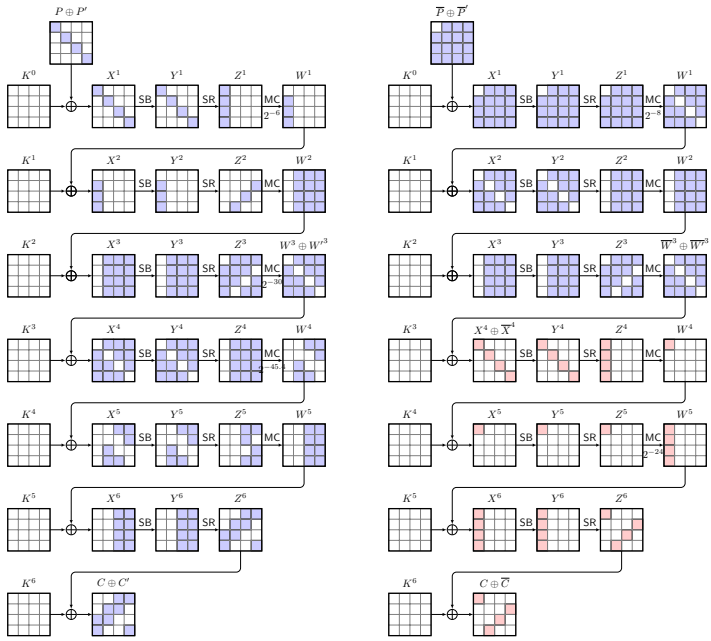


Figure 31: The original boomerang on 6-round `AES`.

prepend one round. Figures 32 and 33 show the upper and lower parts, respectively, of the first truncated-differential trail of pairs of CPs (P, P') to the ciphertexts (C, C'); Figures 34 and 35 then illustrate the trail from $(\overline{C}, \overline{C}') = (C \oplus \Delta C, C' \oplus \Delta C)$ back to their corresponding plaintexts $(\overline{P}, \overline{P}')$. The trail from $P \oplus P'$ to $C \oplus C'$ has a probability of

- ca. 2^{-30} to have an inactive diagonal at ΔW_0^3,
- approximately $4 \cdot 2^{-32 \cdot 4} = 2^{-126}$ to have one inactive slice after Round 8,
- ca. $6 \cdot 2^{-64 \cdot 3} = 2^{-189.4}$ to have two inactive slices after Round 10 that translate into two inactive rows in each state slice differences before the final `MixColumns` operation in Round 13.

Thus, we can add all 2^{128} values to one of the two inactive rows in each state slice to derive $\overline{C}$ and $\overline{C}'$ from C and C', respectively. For simplicity, we choose the topmost row in our descriptions. The backwards trail then has one active state slice at the end of Round 12 and a probability of 2^{-96} to reduce it to a single active row through the inverse Round 11, which leads to a single active slice after eight rounds and mixing. This again leads to a single active column in each slice after Round 8. We aim for that set of columns that was inactive in the top trail in each slice, which guarantees that both pairs have the same trail. Then, it will produce the same inactive diagonal in W^3 as in the forward trail, but in all state slices. Thus, the probability of the truncated trail is approximately $2^{-30-126-189.4-96} = 2^{-441.4}$. Again, if we find one pair with inactive diagonal in all state slices of $\overline{W}^3 \oplus \overline{W'}^3$, by mixing the values of the four rows in the state slices of Z^{13}, it can craft a total of at least eight mixture pairs that all have the same inactive diagonal in all state slices of $\overline{W}^3 \oplus \overline{W'}^3$ to obtain a sufficiently effective filter. We will consider the key bytes of

- the first diagonals of each subkey K_i^1 for all $i \in \{0, 1, 2, 3\}$, that consist of the secret key bytes $SK[0, 5, 10, 15, 16, 21, 26, 31, 32, 37, 42, 47, 48, 53, 58, 63]$ and
- the key K_0^2 that consists of $SK[16..31]$.

Since four bytes overlap, those represent 28 key bytes that can take on 2^{224} values. The

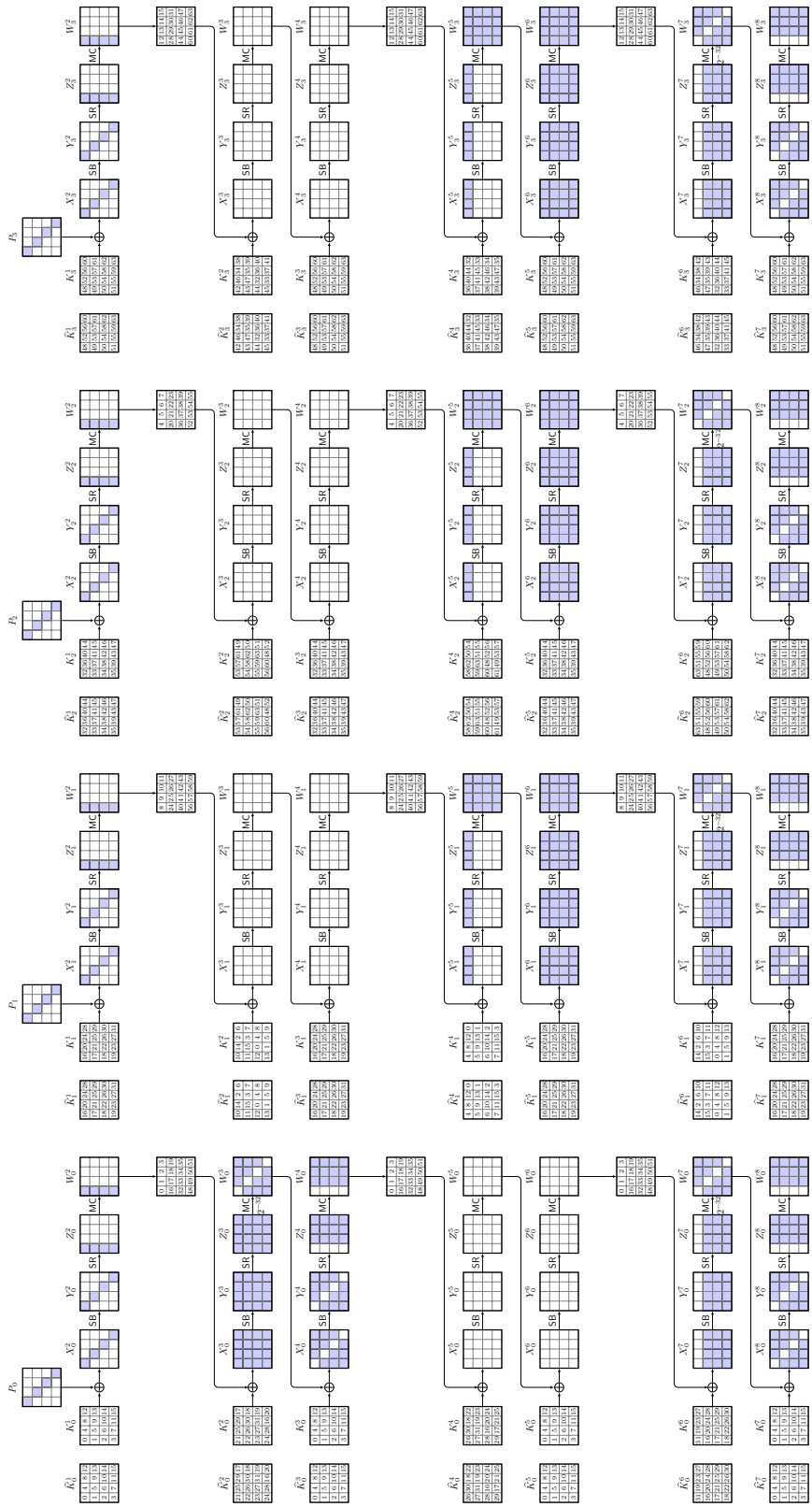


Figure 32: First part of the forward differential trail of the boomerang attack on 12-round Vistrutah-512, from pairs P and P' to intermediate states U.

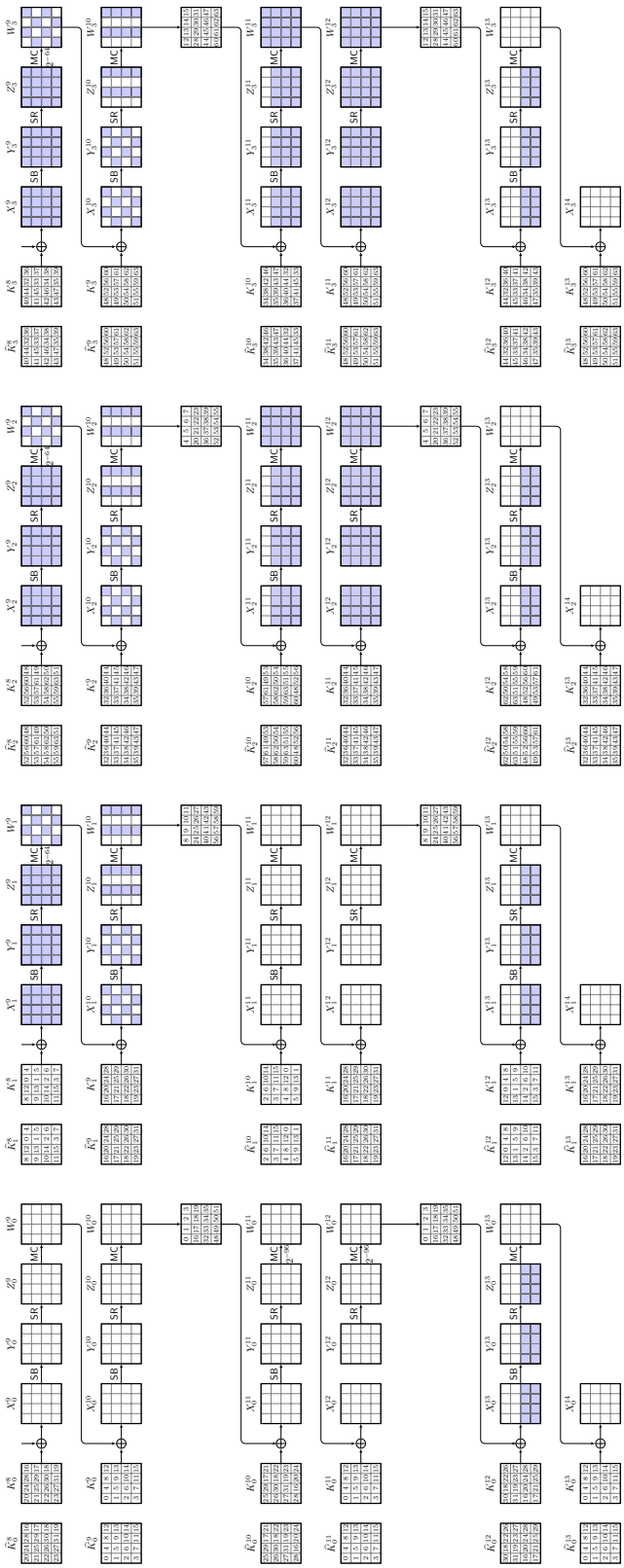


Figure 33: Lower part of the differential trail of the boomerang attack on 12-round `Vistrutah`-512.

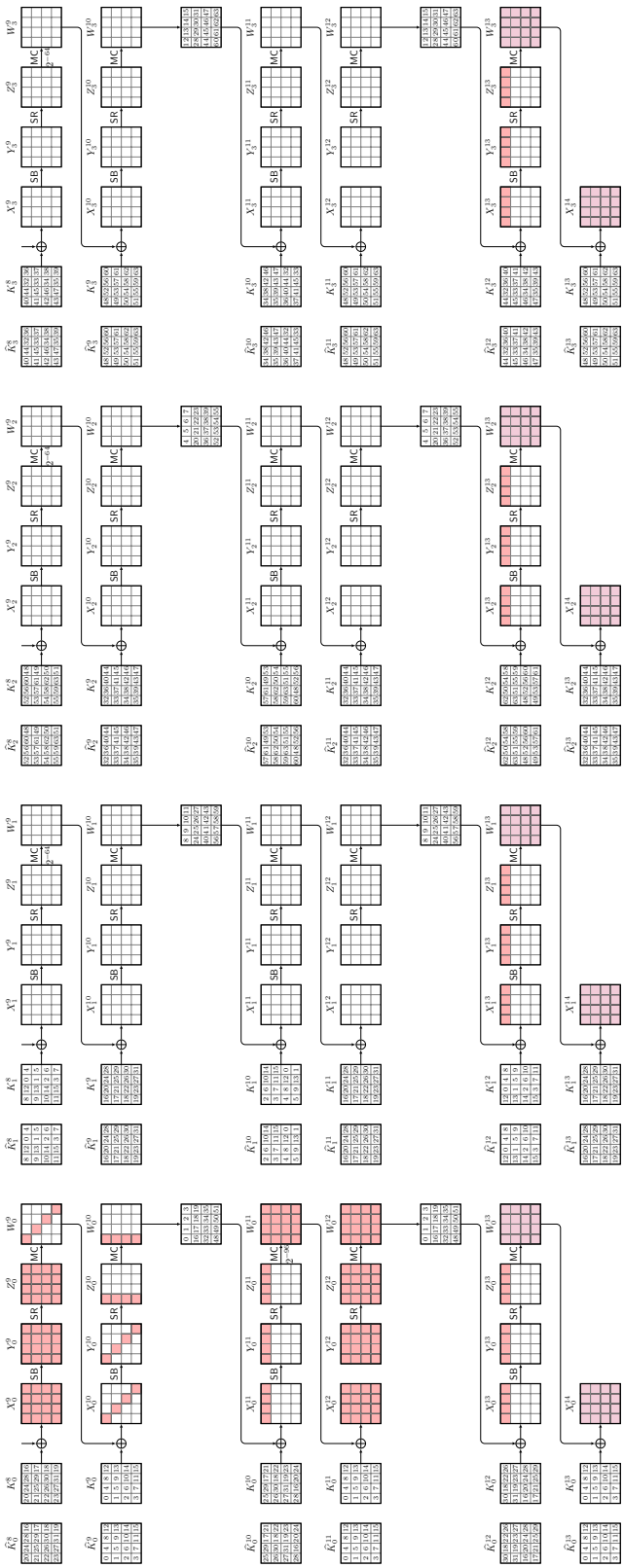


Figure 34: Lower part of the differential trail of the boomerang attack on 12-round Vistrutah-512.

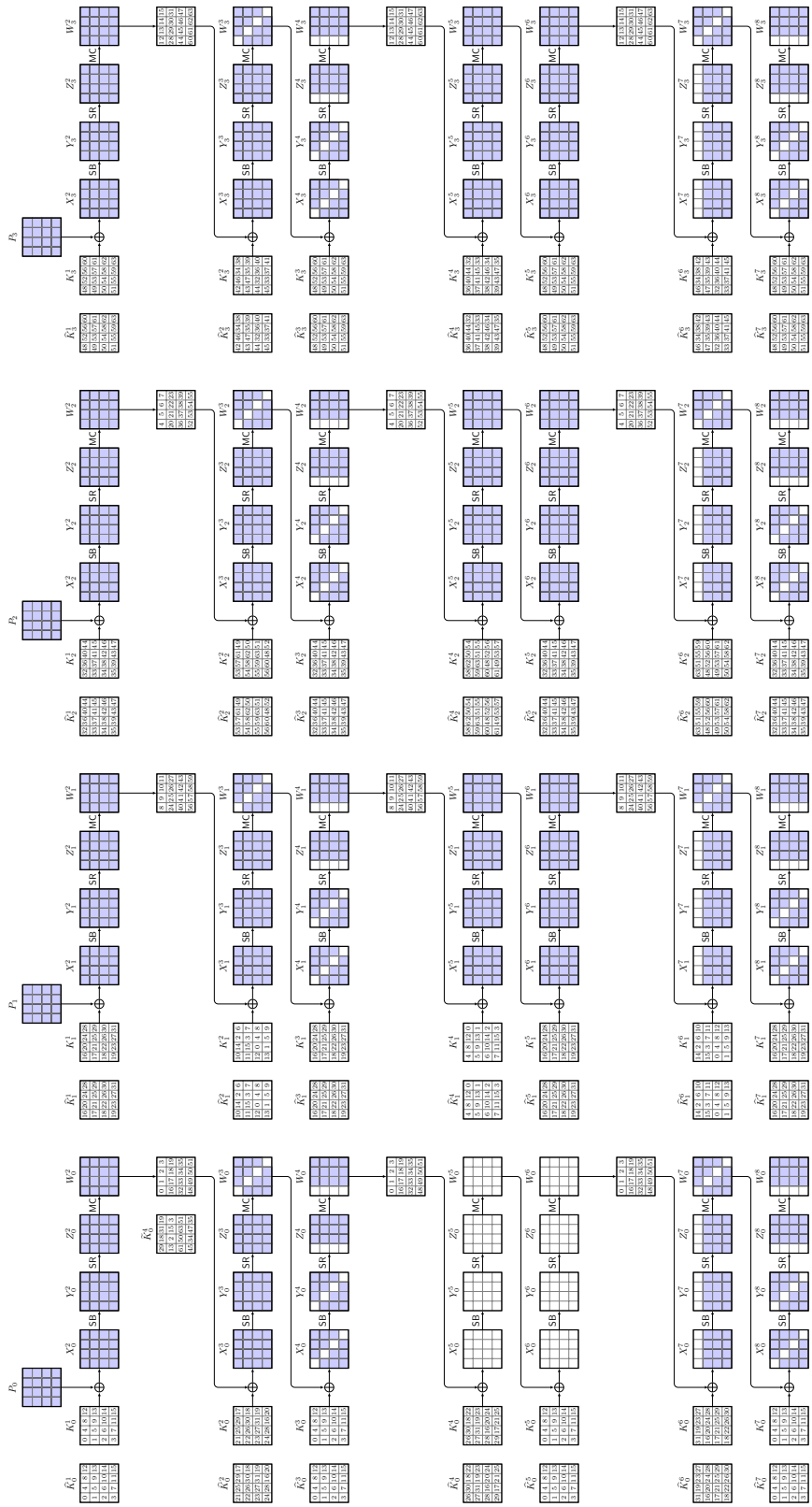


Figure 35: Upper part of the decryption trail of the boomerang attack on 12-round `Vistrutah`-512.

attack steps are as follows.

1. Initialize empty tables $\mathcal{C}$ for all $i \in \{0, \ldots, 5\}$ and a table $\mathcal{K}$ for 2^{224} counters of key candidates.
2. For 2^s times, prepare a distinct structure of 2^{128} plaintexts, iterating over the same diagonal in all four state slices and random (but distinct for all structures) values in the remaining cells.
3. Ask for their corresponding encryptions, invert the final `MixColumns` operation in all slices, and store the resulting states into tables $\mathcal{C}_i$ for $i \in \{0, \ldots, 5\}$ according to their values in two rows, equal for all state slices.
4. For all ciphertext pairs (C, C') from the same structure and that are equal in two state slices:
   - Derive the keys that can lead to an inactive diagonal in ΔW_0^3. We expect $2^{28 \cdot 8} \cdot 4 \cdot 2^{-32} = 2^{194}$ candidates on average.
   - Choose 2^c random values for their values in the first rows in Z^{13} of all state slices and add them to C and C'.
     - For all derived pairs $\overline{C}$ and $\overline{C}'$, ask for their corresponding decryption to $\overline{P}$.
     - Derive seven mixed pairs for each $\overline{C}$ and $\overline{C}'$ by mixing the values of the first row of Z_i^{13} for $i \in \{0, 1, 2, 3\}$ and ask for their corresponding decryptions.
     - For all pairs, derive the keys that suggest to lead to the same inactive diagonal in ΔW_0^3.
     - Mark the key candidates suggested by all mixtures and the plaintext pair.
5. If there exists a key that is suggested eight or more times, use the corresponding mixture pairs to recover the remainder of the key.
6. Test the surviving key candidates with an encryption, and output those that pass.

E.3.2.3 Complexity

From each plaintext structure, we can form $\binom{2^{128}}{2} \approx 2^{255}$ pairs. A partial collision in any two same rows in all slices in the differences of ΔZ^{13}, has a probability of about $6 \cdot 2^{-256} \approx 2^{-253.4}$. Thus, we expect $2^{1.6}$ pairs on average. Since the probability of the first part is approximately $2^{-30-126-189.4} \approx 2^{-345.4}$, we need $2^s = 2^{90.4}$ structures to expect one pair. For that, we expect 2^{92} pairs with a collision in the same two rows in all state slices on average and reencrypt each $2 \cdot 8 \cdot 2^c$ times.

In the backward phase, we have a probability of 2^{-96} to have only a desired row in the active slice of ΔZ^{11} active, which then holds for all pairs of a mixture structure of eight pairs. We take $2^c = 2^{99}$ mixture structures for each ciphertext pair that survives the first filter, which gives on average $2^{99-96} = 2^3$ correct mixture structures that will suggest the same value for 28 key bytes. We also expect $2^{92+99} = 2^{191}$ wrong mixture structures.

For every wrong structure, the plaintext pair suggests $4 \cdot 2^{-32} \cdot 2^{224} = 2^{194}$ key candidates, and each ciphertext pair provides another 32-bit filter. Thus, every mixture structure provides $2^{194} \cdot 2^{-8 \cdot 32} \approx 2^{-62}$ key candidates. For any of the approximately 2^{92} ciphertext pairs, one can expect $2^{99-62} = 2^{37}$ suggested key candidates. However, a correct ciphertext pair will, with significant probability, suggest the same correct key multiple times, eight times on average. Over all 2^{191} mixture structures, we can expect $2^{191-62} = 2^{129}$ suggested key candidates. The probability that the same 224-bit key is suggested by any pair at least eight times is negligibly low. Thus, we expect only the correct key and very few potential false positives. Thereafter, we can continue to find the rest of the key by filtering the keys for the inactive diagonals in the other slices of W^3 for all eight pairs.

The complexity consists of:

- $2^{90.4} \cdot 2^{128} = 2^{218.4}$ encryptions of the same amount of CPs and $2 \cdot 6 \cdot 2^{218.4} \approx 2^{222}$ MAs to store into and read the ciphertexts from the tables,
- $2^{92} \cdot 2^{99} \cdot 2 \cdot 8 = 2^{195}$ decryptions,
- $2^{191} \cdot 8$ key derivations with that amount of MAs to huge tables,
- $2^{191-62} = 2^{129}$ accesses to $\mathcal{K}$ on average,
- a negligible amount of verification of candidates,
- a negligible amount of $2 \cdot 8 \cdot 2^3 \cdot 3 \cdot 4$ key derivations with MAs to huge tables for the other slices of W^3, and
- a negligible amount of full encryptions to test the surviving key candidates.

Thus, the time complexity is dominated by $2^{218.4}+2^{195} \approx 2^{218.4}$ EEs and $2^{222}+2^{194}+2^{129} \approx 2^{222}$ MAs to huge tables. When approximating them with at least one EEs, we obtain at least $2^{218.4}+2^{222} \approx 2^{222.1}$ EEs each. The attack needs memory for 2^{224} key candidates of 28 bytes and $6 \cdot 2^{128} \approx 2^{130.6}$ states at a time, which means approximately $2^{224} \cdot 28/64 \approx 2^{222.8}$ states. In total, the attack needs approximately:

- Data-collection phase: $2^{218.4}$ CPs and 2^{195} ACCs.
- Time: $2^{222.1}$ eight-round EEs.
- Memory: $2^{222.8}$ states.

E.3.3 Boomerang Attack on 12-round `Vistrutah`-512-256

We can mount the 12-round boomerang also on the variant with `Vistrutah`-512-256 and adapt the key-recovery of K^1 as shown in Figure 36.

E.3.3.1 Adapted Attack

The attack considers the key bytes of the first diagonals of each subkey K_i^1 for all $i \in \{0, 1, 2, 3\}$, that consist of the secret key bytes $SK[0, 5, 9, 10, 11, 14, 15, 16, 21, 25, 26, 27, 30, 31]$ and K_0^2 which consists of $SK[16..31]$; those represent 23 bytes of the secret key $SK[0, 5, 9..11, 14..31]$ that can take on 2^{184} values. The attack steps are as follows.

1. Initialize empty tables $\mathcal{C}_i$ for all $i \in \{0, \ldots, 5\}$ and a table $\mathcal{K}$ for 2^{184} counters of key candidates.
2. For 2^s times, prepare a distinct structure of 2^{128} plaintexts, iterating over the same diagonal in all four state slices and random (but distinct for all structures) values in the remaining cells.
3. Ask for their corresponding encryptions, invert the final `MixColumns` operation in all slices, and store the resulting states into tables $\mathcal{C}_i$ for $i \in \{0, \ldots, 5\}$ according to their values in two rows, equal for all state slices.
4. For all ciphertext pairs (C, C') from the same structure and that are equal in two state slices:
   - Derive the keys that can lead to an inactive diagonal in ΔW_0^3. We expect $2^{23 \cdot 8} \cdot 4 \cdot 2^{-32} = 2^{154}$ candidates on average.
   - Choose 2^c random values for their values in the first rows in Z^{13} of all state slices and add them to C and C'.
     - For all derived pairs $\overline{C}$ and $\overline{C}'$, ask for their corresponding decryption to $\overline{P}$.
     - Derive seven mixed pairs for each $\overline{C}$ and $\overline{C}'$ by mixing the values of the first row of Z_i^{13} for $i \in \{0, 1, 2, 3\}$ and ask for their corresponding decryptions.
     - For all pairs, derive the keys that suggest to lead to the same inactive diagonal in ΔW_0^3.
     - Mark the key candidates suggested by all mixtures and the plaintext pair.

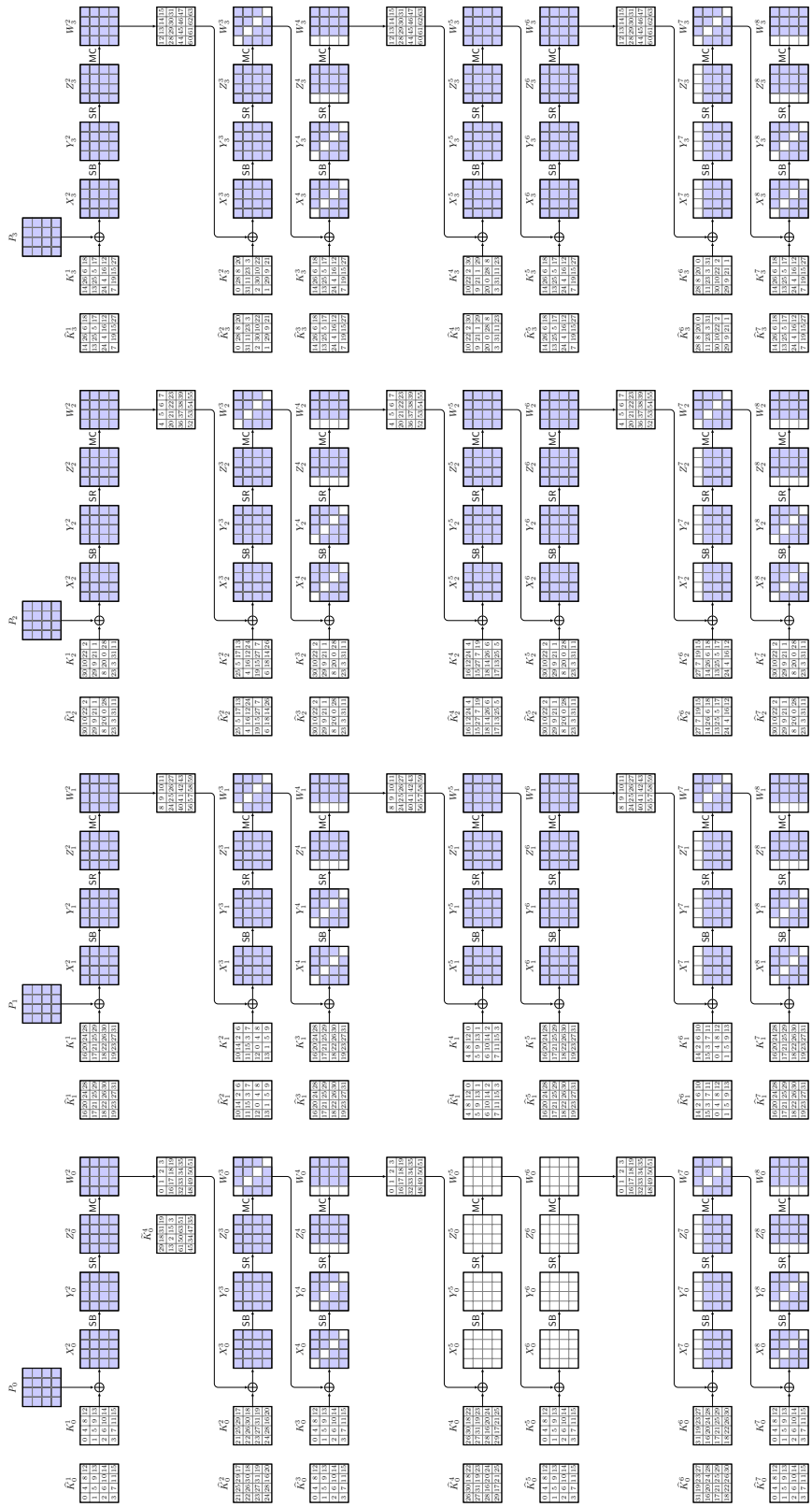


Figure 36: Upper part of the decryption trail used in the boomerang attack on 12-round Vistrutah-512-256.

5. If there exists a key that is suggested eight or more times, use the corresponding mixture pairs to recover the remainder of the key.
6. Test the surviving key candidates, and output the keys that survive.

E.3.3.2 Complexities

The complexity of the attack can be derived similarly as for the boomerang attack on `Vistrutah`-512 in Subsubsection E.3.2. The main difference stems from the fewer involved variables of the key, i.e., 23 instead of 28 key bytes. Since the time complexity remains dominated by the $2^{90.4} \cdot 2^{128} = 2^{218.4}$ encryptions of the same amount of CPs and $2 \cdot 6 \cdot 2^{218.4} \approx 2^{222}$ MAs to store into and read the ciphertexts from the tables as before, it does not change noticably. However, the attack needs less memory for 2^{184} key candidates of 23 bytes instead of 2^{224} before and $6 \cdot 2^{128} \approx 2^{130.6}$ states at a time. Thus, the memory complexity reduces to approximately $2^{184} \cdot 23/32 \approx 2^{183.5}$ states. In total, the attack needs approximately:

- Data-collection phase: $2^{218.4}$ CPs and 2^{195} ACCs.
- Time: $2^{222.1}$ eight-round EEs.
- Memory: $2^{183.5}$ states.

E.4 Mixture-differential Cryptanalysis

E.4.1 Mixture Attack on 8-round Vistrutah-256

In the following, we mount a mixture attack on eight rounds of Vistrutah-256.

E.4.1.1 Distinguisher

The differential trail, with active cells in blue, red, and purple, and the key-recovery part, with active cells in yellow, is shown in Figure 37. The differential is probabilistic:

- in Round 2, with the transition $\Delta Z_i^2 \to \Delta W_i^2$ having a probability of 2^{-64} for each slice and
- in Round 5, with the transition $\Delta Z_i^5 \to \Delta W_i^5$ having a probability of 2^{-96} for each slice.

Therefore, for one pair, a position-fixed truncated-differential trail has a probability of approximately $2^{-(64+96)\cdot 2} = 2^{-320}$. However, for a pair that follows the trail, we can exchange one slice to form a mixture pair, e.g. construct $(\overline{P}, \overline{P}')$ from a pair (P, P') by defining $\overline{P} = (P_0, P_1')$ and $\overline{P}' = (P_0', P_1)$. Since each slice difference in the second step depends only on the difference in one slice from the first step, and since the corresponding parts do not influence each other until W_0^5 and W_1^5, the mixture pair also follows the truncated-differential path until the end with conditional probability 1.

We consider two potential trails at the end of Round 2 (the active ws in ΔW_0^2 and ΔW_1^2 may be flipped); moreover, there are four possible options for a single active diagonal in ΔW_0^5, which define what diagonal we want active in ΔW_1^5. Thus, we consider a set of $2 \cdot 4$ trails that has probability ca. 2^{-317}. Moreover, in the final attack, one quartet of two mixed pairs will provide us with 2^{64} candidates for 128 key bits by decrypting the two active anti-diagonals in ΔZ_0^8 and ΔZ_1^8 each, that are marked yellow in Figure 37 by one round for both pairs. By waiting for a second pair that covers the inactive anti-diagonals in ΔZ^8, we can Since a first pair defines what anti-diagonals a second mixture quartet should be active and inactive in, there are no four options for the second quartet at ΔW^5. Thus, such a second pair has probability 2^{-319} to occur, and both have a probability of approximately $(1 - (1 - 2^{-317})^{2^{319}}) \times (1 - (1 - 2^{-319})^{2^{319}}) \approx 0.62$ to occur both when we will collect 2^{319} pairs that are assumed to be random. For a random permutation, the probability

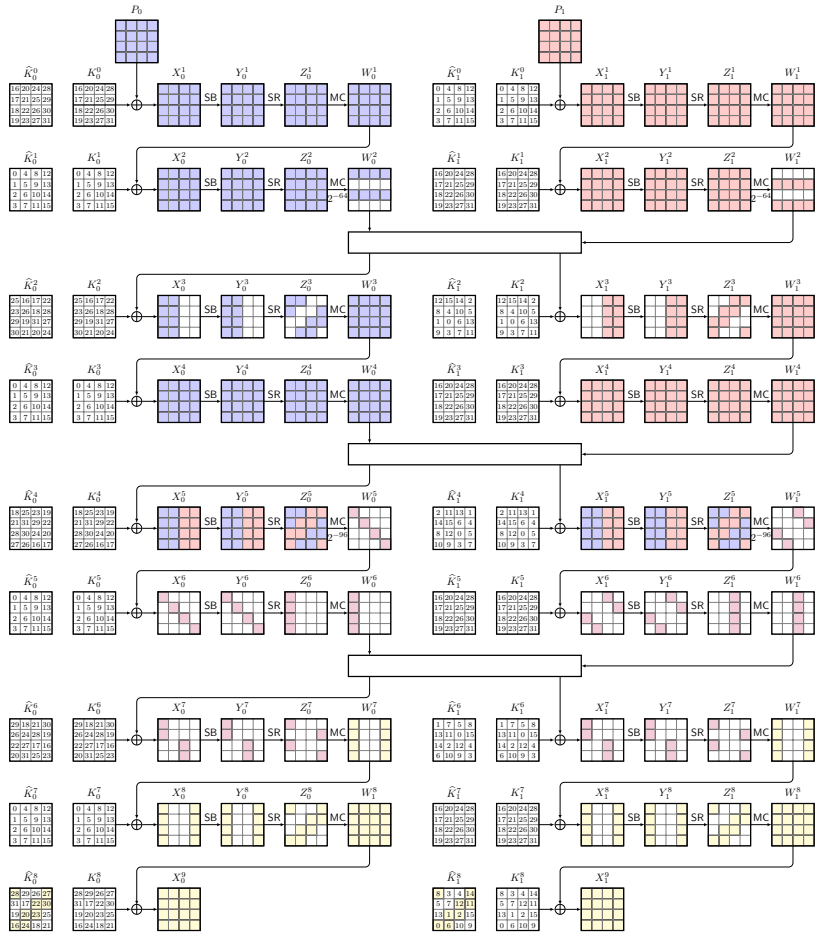


Figure 37: The mixture attack on 8-round Vistrutah-256.

is approximately $4 \cdot 2^{-128 \cdot 2} = 2^{-254}$. Thus, we can expect approximately $2^{319-254} = 2^{65}$ quartets. From each active anti-diagonal of ΔZ_i^8 to a column with two active bytes in ΔZ_i^7, we expect $2^{-16-16} \cdot 2^{32} \approx 1$ key candidate. Thus, we expect approximately 2^{65} candidates for 128 key bits. On average, we expect 2^{63} quartets for each of the four options of active anti-diagonals in ΔZ^8. However, they may be distributed in equal shares in two combinable options, so that at most $2^{64} \cdot 2^{64} = 2^{128}$ pairs that suggest one key candidate for all 256 key bits each are possible.

E.4.1.2 Attack

The attack steps are as follows:

1. Initialize four empty hash tables $\mathcal{C}_i$ for $i \in \{0, 1, 2, 3\}$ and four tables $\mathcal{M}_i$.
2. Initialize two hash tables $\mathcal{H}_j$, $j \in \{0, 1\}$, where each table takes a four-byte difference in an anti-diagonal in ΔZ^8 and returns all values after the key addition with the `MixColumns`-inverted key such that only Bytes 0 and 2 are active in the difference of the corresponding column in ΔZ^8 for $\mathcal{H}_0$ and only Bytes 1 and 3 are active for $\mathcal{H}_1$.
3. Choose two sets of 2^{80} random 128-bit values and combine them to 2^{160} plaintexts.
4. Ask for their corresponding encryptions and invert their final `MixColumns` operation. Store the resulting values into all tables $\mathcal{C}_i$ according to the values in one out of four

sets of anti-diagonals of $Z_0^8 \| Z_1^8$.

5. Identify pairs in all four tables $\mathcal{C}_i$. We expect $4 \cdot 2^{319} \cdot 2^{-128} = 2^{193}$ pairs.
6. For each pair, identify its mixture and lookup whether it also is in the same cell in $\mathcal{C}_i$, which has probability 2^{-128}. We expect $2^{193-128} = 2^{65}$ mixture quartets.
7. For each mixture quartet, lookup with $\mathcal{H}_0$ and $\mathcal{H}_1$ the key candidates it suggests. Store each quartet into either $\mathcal{M}_i$ depending on their active anti-diagonals in ΔZ^8.
8. For all combinations of quartets in $\mathcal{M}_0$ and $\mathcal{M}_2$, test their combined now full-key candidate with one encryption. Similarly, for all combinations of quartets in $\mathcal{M}_1$ and $\mathcal{M}_3$, test their combined now full-key candidate with one encryption.
9. Output the keys that survive this test.

E.4.1.3 Complexity

The computational efforts consist of

- 2^{160} encryptions
- $8 \cdot 2^{160}$ MAs for finding pairs.
- $2^{193} \cdot 2$ MAs for finding quartets.
- $2 \cdot 2^{65}$ MAs for storing quartets.
- At most $2^{128} \cdot 2 \cdot 2 \cdot 8 = 2^{133}$ MAs to find temporary key candidates.
- $2^{128} \cdot 16 \cdot 2^{16} = 2^{147}$ MAs to find overlapping key candidates for each column in both quartets.
- 2^{128} encryptions to test key candidates.

This yields $2^{160} + 2^{128} \approx 2^{160}$ EEs and $2^{163} + 2^{194} + 2^{66} + 2^{133} + 2^{147} \approx 2^{194}$ MAs to huge tables. Approximating the accesses to tables with one full encryption, we obtain approximately 2^{194} EEs. The attack requires storage for $2 \cdot 4 \cdot 2^{160} = 2^{163}$ plaintext and ciphertext states at a time, and additional memory for hash tables and quartets that are not significant in the overall memory complexity.

- Computations: 2^{160} EEs and 2^{194} MAs.
- Data: 2^{160} CPs.
- Memory: 2^{163} states.

E.4.2 Mixture Distinguisher on 10-round `Vistrutah`-512

We obtain basically the same mixture distinguisher also for `Vistrutah`-512, here from Rounds 2–11.

E.4.2.1 Trails

The distinguisher is shown for Rounds 2–6 in Figure 38 and for Rounds 7–11 in Figure 39. we choose plaintexts from the set of the first two active diagonals in every plaintext's slices $P_0, \ldots, P_3$. In total, the distinguisher then searches for pairs that follow a certain truncated-differential trail as shown in the figures that needs the following events:

- Round 3: ΔW_0^3 is active only in a single diagonal and ΔW_1^3 is inactive in the same diagonal, which has probability approximately $4 \cdot 2^{-96} \cdot 2^{-32} = 2^{-126}$.
- Round 7: $\Delta W_i^7 \in \mathcal{D}_j$, for all $i, j \in \{0, 1, 2, 3\}$, which has probability approximately $4 \cdot 2^{-32 \cdot 4} = 2^{-126}$.

The approximated probability for the truncated-differential trail is then $2^{-126-126} = 2^{-252}$. If those events occur, the bytes in one row, which must be the same for all slices, will be inactive in all state slices ΔZ_i^{11}. At random, this event will occur for a pair with probability of ca. $4 \cdot 2^{-128} = 2^{-126}$. However, we can construct a second mixture pair

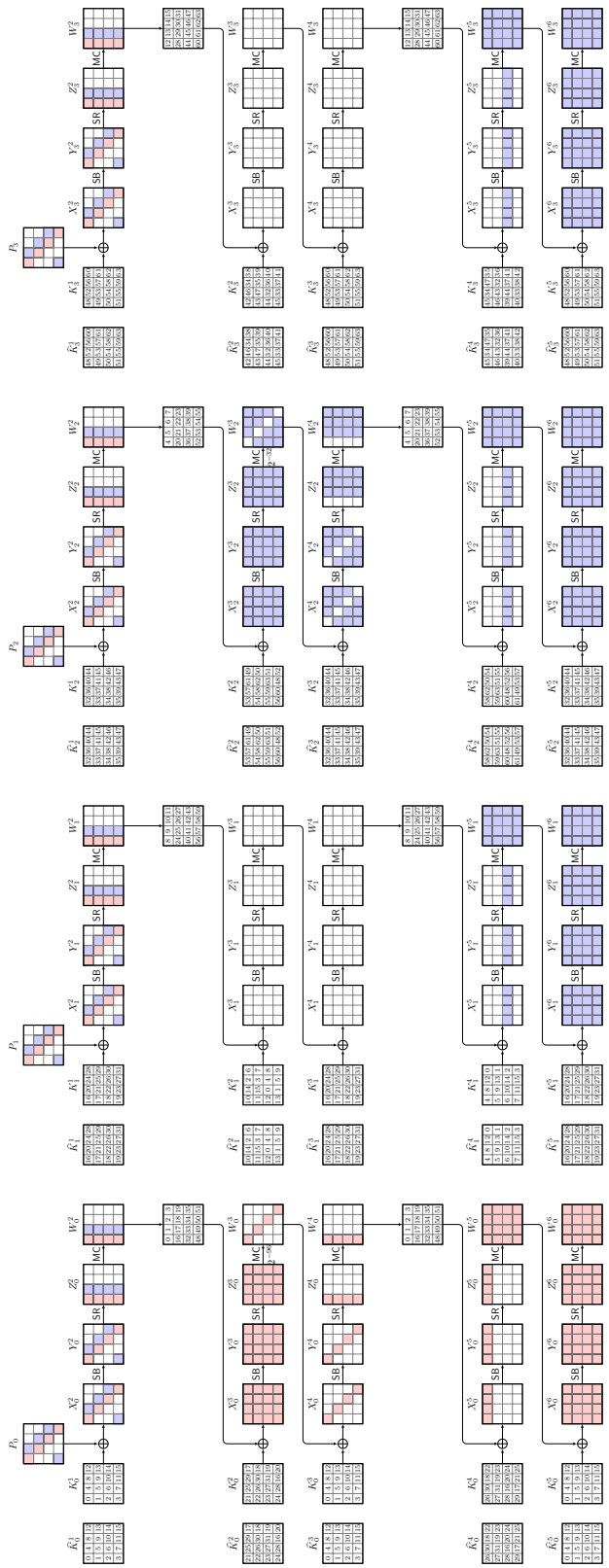


Figure 38: Mixture distinguisher on 10-round Vistrutah-512: Rounds 2–6.

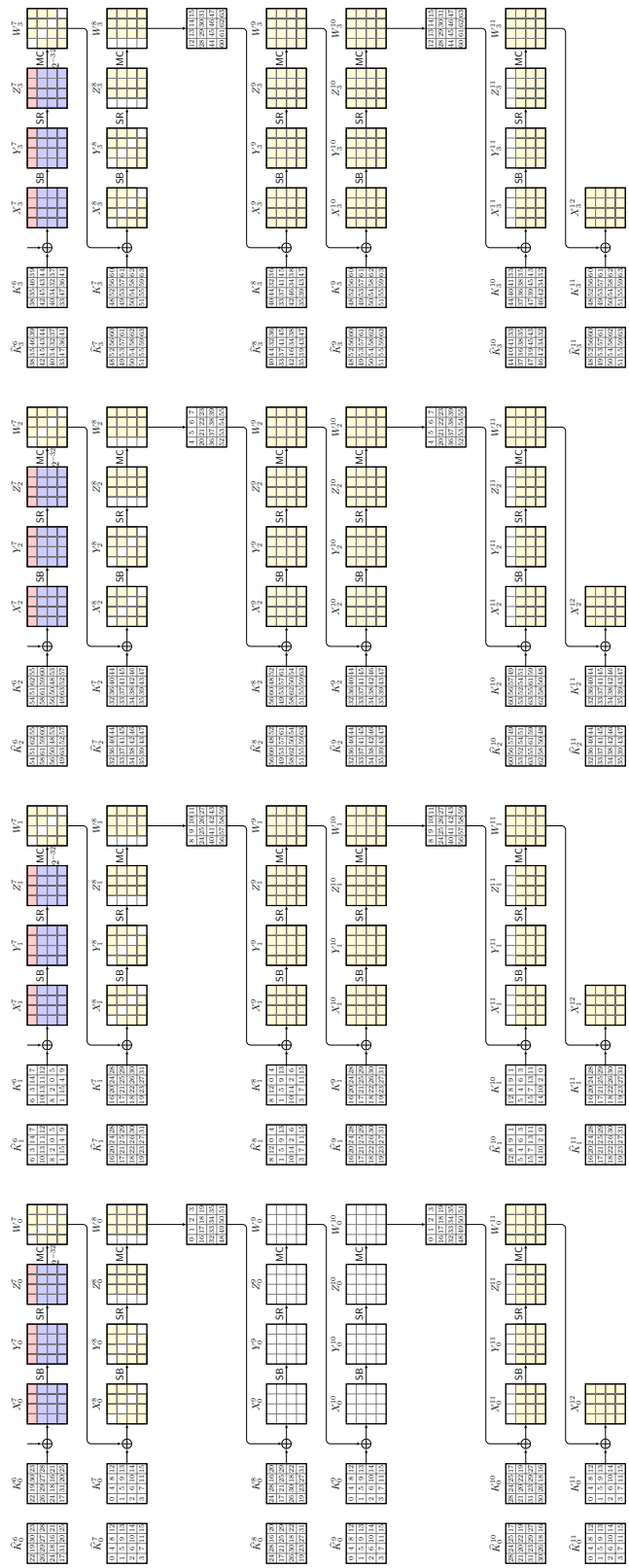


Figure 39: Mixture distinguisher on 10-round `Vistrutah`-512: Rounds 7–11.

$(\overline{P}, \overline{P}')$, constructed from a pair (P, P') by swapping the red and parts of P and P' in Figure 38. This means, $\overline{P}$ will take on the (blue) values from P in the second diagonal but the (red) values from P' in the first diagonal of each slice. Analogously, $\overline{P}'$ will take on the (blue) values from P' in the second diagonal but the (red) values from P in the first diagonal of each slice. A so-mixed pair will have the same event as above, to have the same inactive row that is the same in all slices of ΔZ^{11}, with conditional probability one for the real 10-round Vistrutah-512. Therefore, the probability that this event happens for a pair and its mixture pair is approximately 2^{-252} for the real 10-round construction whereas it is approximately 2^{-254} for a random 512-bit permutation.

E.4.2.2 Distinguisher

The distinguishing procedure is the following:

1. Initialize four hash tables $\mathcal{C}_i$ for $i \in \{0, 1, 2, 3\}$ for the state rows.
2. Choose 2^{127} plaintexts iterating over the values in the first two plaintext diagonals of all plaintext slices and leaving all other diagonals the same (but random) for all plaintexts.
3. Ask for their corresponding ciphertexts, invert the final MixColumns operation, and store each resulting text into each table $\mathcal{C}_i$ according to their combined values in the i-th row in each state combined over all state slices.
4. Identify matching pairs from all tables $\mathcal{C}_i$; for each matching pair, construct a mixture plaintext pair from exchanging their red and blue diagonals in the plaintexts slices as described above. If it has not already been queried before, ask for its corresponding ciphertexts, and invert the final MixColumns operation for them.
5. If there exists a mixture pair whose difference is inactive in at least one row over all slices, output real.
6. Output random otherwise.

From 2^{127} plaintexts, we can form approximately 2^{253} pairs, out of which we can expect $2^{253} \cdot 2^{-126} = 2^{127}$ to satisfy the first filter of their ciphertext difference. Among them, we can expect $2^{253} \cdot 2^{-252} = 2^1$ correct pair(s). For each of the 2^{127} surviving pairs, we construct a mixture pair and asks for its encryption. We can expect $2^{127} + 2 \cdot 2^{127} \approx 2^{128.6}$ CPs to suffice.

Furthermore, we can expect approximately $2^{127} \cdot 2^{-128} = 2^{-1}$ false positives, a few wrong pairs with a very high probability for a random permutation. In contrast, we can expect a good mixture pair with probability greater than $1 - e^{-1}$. The complexity consists of $2^{127} + 2 \cdot 2^{127} \approx 2^{128.6}$ encryptions, $2 \cdot 4 \cdot 2^{127} = 2^{130}$ MAs to huge tables, $2 \cdot 2^{127} = 2^{128}$ XORs, and $(2^{127} + 2 \cdot 2^{127}) \cdot 1/10 \approx 2^{125.3}$ EEs for inverting the final MixColumns operations.

- Computations: $2^{128.6} + 2^{125.3} \approx 2^{128.7}$ EEs and 2^{130} MAs to huge tables. If we approximate MAs by (at least) one EE, we obtain $2^{128.7} + 2^{130} \approx 2^{130.5}$ EEs
- Data: At most $2^{128.6}$ CPs.
- Memory: $4 \cdot 2 \cdot 2^{127} = 2^{130}$ states.

E.4.3 Mixture Distinguisher on 12-round Vistrutah-512

In the following, we can adapt the six-round exchange (or mixture) distinguisher by Bardeh and Rønjom [BR19] to 12 rounds of Vistrutah-512, from Rounds 2–13.

E.4.3.1 Distinguisher

Rounds 2–8 of the distinguisher are displayed in Figure 40 and Rounds 9–13 in Figure 41. We choose plaintexts from the set of three active diagonals in all plaintext slices $P_0, \ldots,$

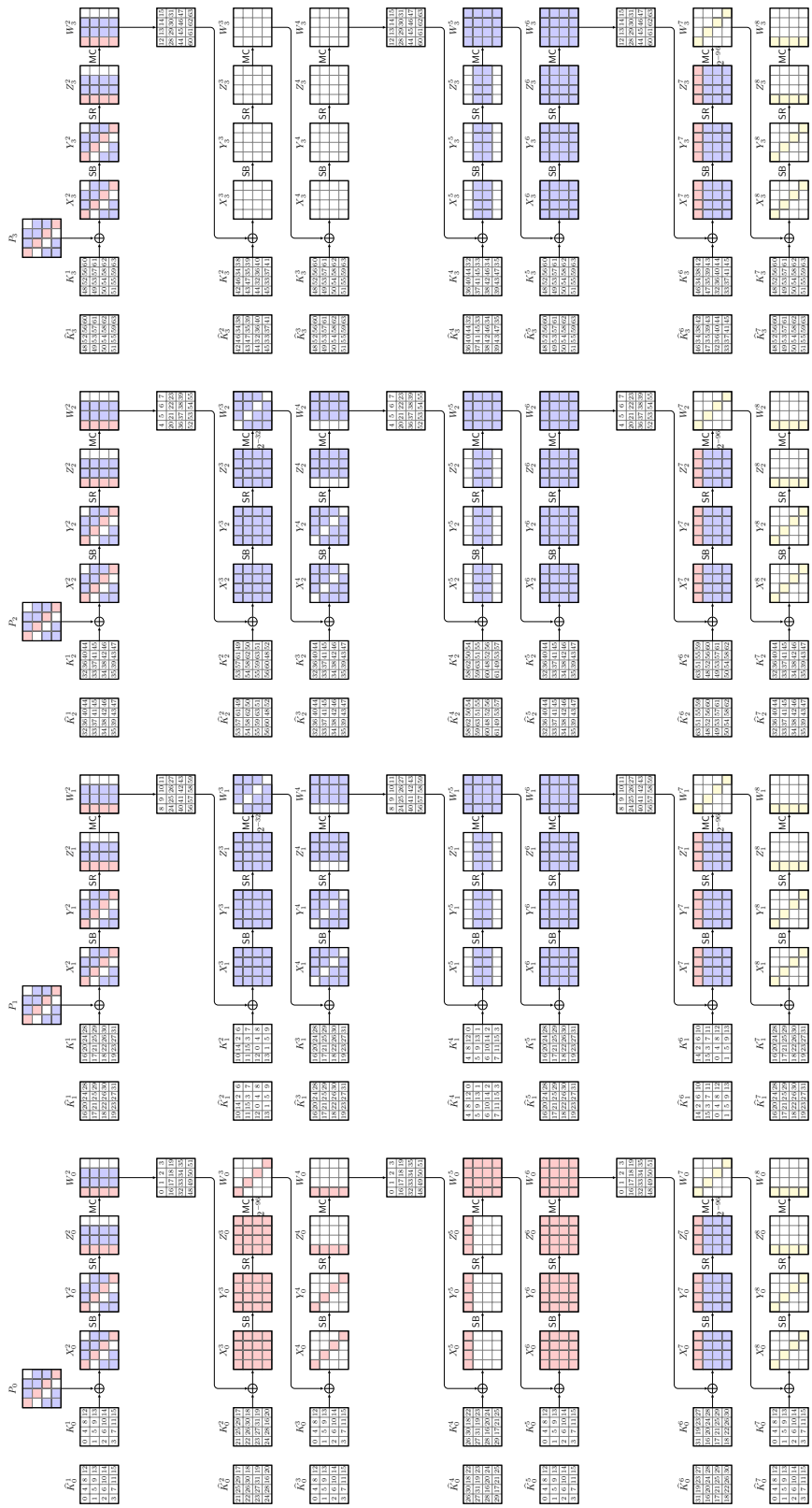


Figure 40: Mixture distinguisher on 12-round `Vistrutah`-512: Rounds 2–8.

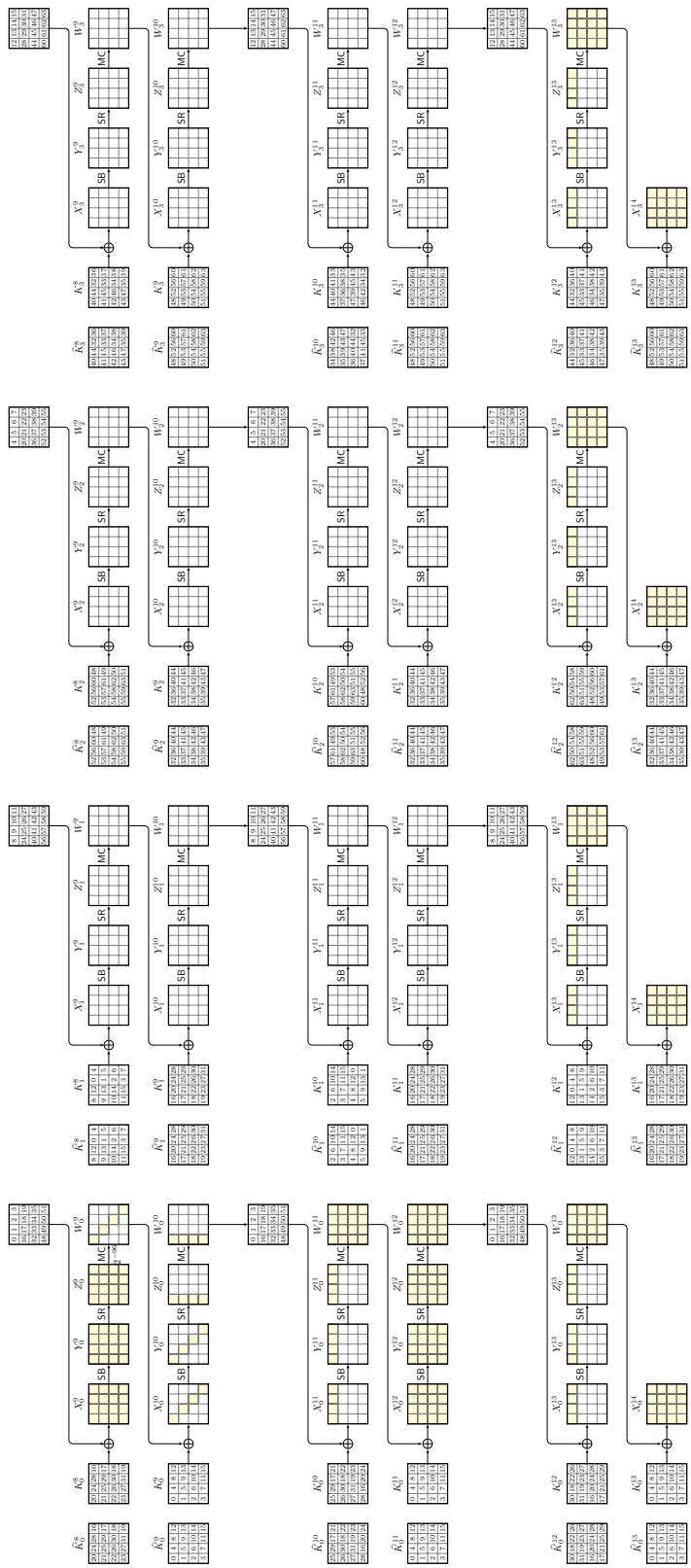


Figure 41: Mixture distinguisher on 12-round Vistrutah-512: Rounds 9–13.

P_3. Using the distinguisher we search for pairs that follow a certain truncated-differential trail as shown in the figures, waiting for a set of events:

- Round 3: $\Delta W_0^3 \in \mathcal{D}_0$ and $\Delta W_1^3, \Delta W_2^3 \in \mathcal{D}_{1,2,3}$, which has probability approximately $2^{-96} \cdot 2^{-32} \cdot 2^{-32} = 2^{-160}$. Given that one can choose four possible diagonals, we obtain 2^{-158}.
- Round 7: $\Delta W_i^7 \in \mathcal{D}_0$, for all $i \in \{0, 1, 2, 3\}$, which has probability approximately $2^{-96 \cdot 4} = 2^{-384}$. Again, there are four possible trails, yielding 2^{-382}.
- Round 9: $\Delta W_0^9 \in \mathcal{D}_0$, with probability approximately 2^{-96}. Again, there are four possible trails, yielding 2^{-94}.

The approximated probability for a fixed truncated-differential trail is $2^{-160-384-96} = 2^{-640}$ and 2^{-634} when considering 4^3 trail options. If those events occur, the bytes in at most one row (the same for all slices) will be active in all state slices ΔZ_i^{13}. Standalone, this event occurs with probability $4 \cdot 2^{-384} = 2^{-382}$. However, a second dependent pair is obtained with amplified probability.

Given a set of pairs that fulfill the output difference, we then form a mixture pair from each of them by exchanging the first diagonal values in each plaintext slice P_i. If the original pair fulfills the truncated differential, the mixed pair is guaranteed to fulfill the trail through the two Steps covering Rounds 3–6 and its wrapping Rounds 2 and 7 to ΔX_i^8 of all state slices. Thus, it satisfies $\Delta W_i^7 \in \mathcal{D}_0$, for all $i \in \{0, 1, 2, 3\}$, with probability one, and has only probability 2^{-96} to also have $\Delta W_0^9 \in \mathcal{D}_0$ and probability 2^{-94} for any single diagonal ΔW_0^9 that influences which row of state slices will be the only active one in the slices ΔZ_i^{13}. As a result, the probability for a pair and mixture pair to both follow the differential is approximately $2^{-634} \cdot 2^{-94} = 2^{-728}$. In contrast, the probability that a pair and a so-mixed pair will have only one state row active each in the slices ΔZ_i^{13} is approximately $(4 \cdot 2^{-384})^2 = 2^{-764}$.

The procedure is the following:

1. Initialize four hash tables $\mathcal{C}_i$ for $i \in \{0, 1, 2, 3\}$ for the state rows.
2. Choose 2^{365} CPs iterating (randomly) over values in the first three diagonals of all plaintext slices.
3. Ask for their corresponding ciphertexts, invert the final `MixColumns` operation, and store each resulting text into each table $\mathcal{C}_i$ according to their value in the i-th row combined over all state slices.
4. Identify pairs from $\mathcal{C}_i$, construct a mixture plaintext pair from exchanging their first diagonal in each plaintext slice and ask for its corresponding encryption, and invert the final `MixColumns` operation.
5. If there exists a mixture pair whose difference is active in at most a single state row over all slices, output "real," else output "random."

E.4.3.2 Complexity

From 2^{365} plaintexts, we can form approximately 2^{729} pairs, out of which we can expect $2^{729} \cdot 4 \cdot 2^{-384} = 2^{347}$ to satisfy the first filter of their ciphertext difference. Among them, we can expect $2^{729} \cdot 2^{-634} = 2^{95}$ correct pairs. For each of the 2^{347} pairs, we construct a mixture pair and asks for its encryption. We can expect approximately $2^{347} \cdot 4 \cdot 2^{-382} = 2^{-33}$ so no wrong pairs with high probability for a random permutation. In contrast, we can expect a good mixture pair with probability $> 1 - e^{-1}$ since we have 2^{95} correct pairs and a probability of approximately 2^{-94} for a mixture pair to fulfill the trail. The complexity consists of

- Computations: $2^{365} + 2 \cdot 2^{347} \approx 2^{365}$ encryptions, $2 \cdot 4 \cdot 2^{365} = 2^{368}$ MAs to huge tables, $2 \cdot 2^{347}$ XORs, and $(2^{365} + 2 \cdot 2^{347}) \cdot 1/12 \approx 2^{361.4}$ EEs for inverting the final

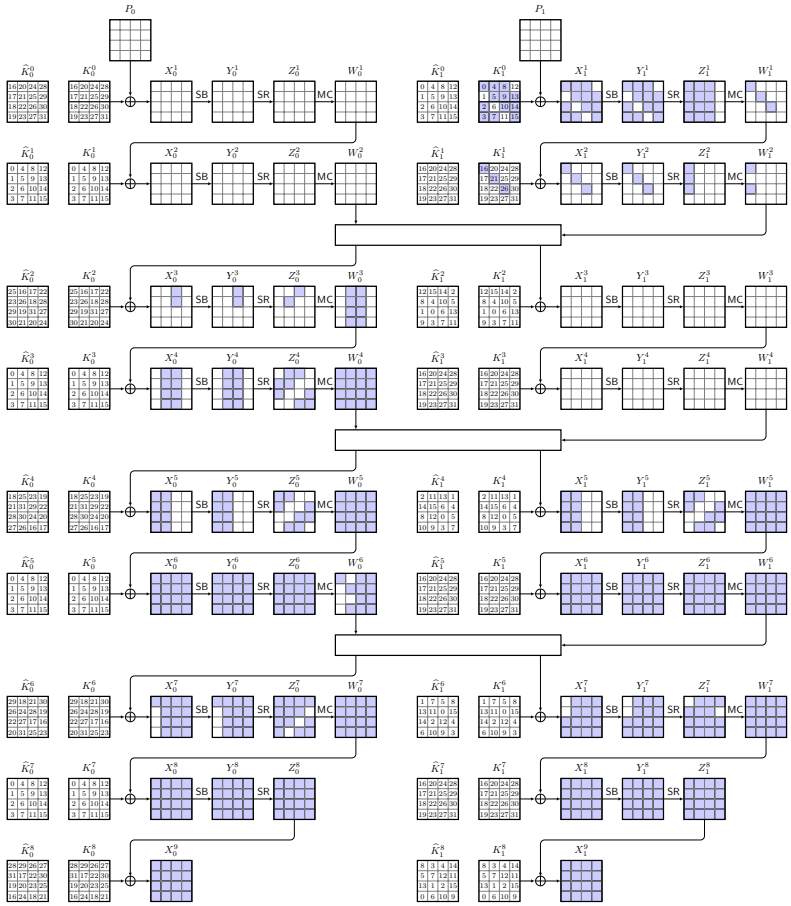


Figure 42: Forward differential trail of the yoyo attack from text pairs P and P' to C and C' through 8-round Vistrutah-256.

MixColumns operations. If we approximate MAs by at least one full encryption, we obtain $2^{368.2}$ EEs.

- Data: $2^{365} + 2^{348} \approx 2^{365}$ CPs.
- Memory: $4 \cdot 2^{365}$ plaintext-ciphertext pairs, i.e., 2^{368} states.

E.5 Yoyo Cryptanalysis

E.5.1 Yoyo Attack on 8-round Vistrutah-256

Since the high-level structure of Vistrutah-256 differs significantly from an upscaled version of the AES, mounting an analogue of the yoyo attack on eight rounds needs more tailoring and higher complexity than for the 512-bit version. The for- and backward trails are illustrated in Figures 42 and 43, respectively.

E.5.1.1 Data-collection Phase

The forward trail is depicted in Figure 42. The steps for constructing and encrypting the forward pairs are as follows:

1. Initialize a hash table $\mathcal{H}_1$ which, given an diagonal input difference in $(\mathbb{F}_{2^8})^4$, returns all approximately 2^8 keys on average that lead to a difference in only the first byte

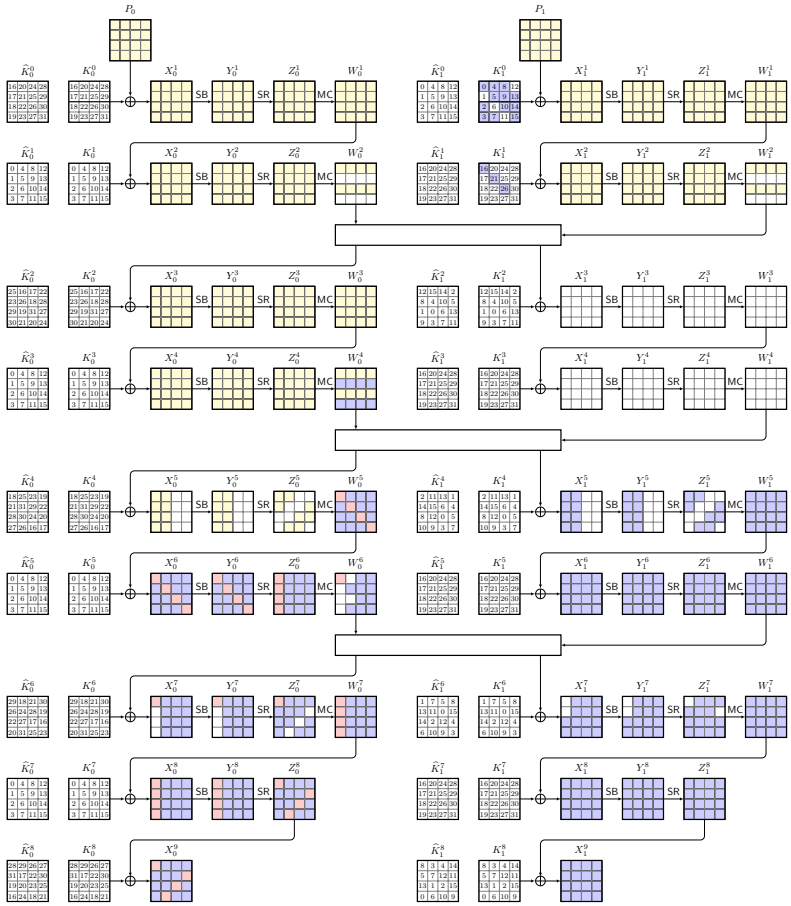


Figure 43: Backward differential trail of the yoyo attack on 8-round `Vistrutah`-256. The exchanged part is marked in red, the active (or unknown) byte differences that likely differ from that of the forward trail are marked in yellow.

after one AES round.

2. Initialize another hash table $\mathcal{H}_2$ that yields the 2^8 keys on average that, given a difference $(\alpha_0, \alpha_1, \alpha_2, \alpha_3) \in (\mathbb{F}_{2^8})^4$, will produce a difference $(\beta_0, 0, \beta_2, 0) \in (\mathbb{F}_{2^8})^4$ after one AES round.
3. Initialize hash tables $\mathcal{K}$, $\mathcal{C}$, and $\mathcal{P}$.
4. Combine twelve sets of random values from $\mathbb{F}_{2^8}$ with s values each. Those sets will be used to form the bytes of the first three diagonals of $\mathcal{D}_{0,1,2}$ of P_1. Choose the values of the remaining bytes randomly to form s^{12} plaintexts.
5. Ask for their corresponding encryptions after eight rounds and store them into $\mathcal{C}$.
6. For all s^{12} ciphertexts C and all $i \in \{0, \ldots, 2^{32} - 1\}$:
   (a) Derive a ciphertext C^i by replacing its inverse diagonal $\mathcal{ID}_0(C_0)$ with the unique $4 \cdot 8$-bit encoding that corresponds to the current value of i.
   (b) Ask for the corresponding decryption of C^i to P^i from the oracle and store the plaintexts in $\mathcal{P}$.

E.5.1.2 Attack

The steps for forming and encrypting the forward pairs are as follows:

1. Sort the $n_p = s^{12} \cdot (s-1)^{12}/2$ plaintext-and-ciphertext pairs $((P, C), (P', C'))$ such that their corresponding plaintexts P and P' have twelve active bytes in $\mathcal{D}_{0,1,2}$ into $n_m = n_p/4$ pairwise distinct mixture sets of 4 pairs each, such that the four pairs in each set have the same differences and are defined by exchanging active diagonals.
2. For all n_p pairs $((P, C), (P', C'))$:
   (a) Look up the output-exchanged pair $((P^{(2)}, C^{(2)}), (P^{(3)}, C^{(3)}))$ using $\mathcal{C}$ and $\mathcal{P}$.
   (b) Look up the diagonal differences of $\mathcal{D}_i(P_1 \oplus P_1')$ for $i \in \{0, 1, 2\}$ in $\mathcal{H}_1$ (the differences for the diagonals 1 and 2 must be rotated before lookup) to find 2^8 key candidates for K_1^0 on average for each diagonal. This yields SK[0, 2, 3, 4, 5, 7, 8, 9, 10, 13, 14, 15].
   (c) For each key candidate, look up the 2^8 possible keys K_1^1, corresponding to $SK[16, 21, 26]$, that lead to a difference $(\beta_0, 0, \beta_2, 0)$ after Round 2 in the leftmost column.
   (d) Encrypt the second plaintext pair $P^{(2)}$ and $P^{(3)}$ partially up to Z^2[0..3] with the key candidate. This yields an eight-bit filter; proceed only for those candidates that fulfill it.
   (e) With the second pair's $\Delta X_1^2[3, 4, 9]$, look up in $\mathcal{H}_2$ the 2^{16} candidates for $SK[20, 25, 19]$.
   (f) With the second pair's $\Delta X_1^2[2, 7, 8]$, look up in $\mathcal{H}_2$ the 2^{16} candidates for $SK[24, 18, 23]$.
   (g) With the second pair's $\Delta X_1^2[1, 6, 11]$, look up in $\mathcal{H}_2$ the 2^{16} candidates for $SK[17, 22, 27]$. At that stage, also SK[16..27] are known.
   (h) For the first pair, partially decrypt from ΔC_0 to ΔZ_0^7[4..15], given K_0^8[2, 3, 5..12, 14, 15]. This yields an 16-bit filter for two of the inactive difference bytes in $\Delta Z_0^7[10, 13]$. Thereupon, we get 2^8 candidates for $K_0^8[1, 4]$ that correspond to $SK[29, 30]$ that satisfy the inactive byte in $\Delta Z_0^7[7]$.
   (i) For the second pair from ΔP_0 to ΔW_0^2[0..11], guess $SK[28]$ to derive Y^2[0, 2, 3, 5..10], which yields a 24-bit filter, so 2^{-16}.
   (j) For the second pair guess the 2^8 candidates for $SK[30]$ and derive ΔY_0^2[0..11], which yields another 24-bit filter to have the correct differences in $\Delta Y^2[1, 4, 7]$ from the previous step. At this point, the full SK[16..31] is known.
   (k) For 2^{16} candidates $SK[1, 12]$, compute the first pair from ΔC_1 to ΔZ_1^7 and then use the 8-bit filter for the inactive byte $\Delta Z_1^7[13]$.
   (l) Similarly, for 2^{16} candidates $SK[6, 11]$, compute the first pair from ΔC_1 to ΔZ_1^7 and then use the 8-bit filter for the inactive byte $\Delta Z_1^7[0]$.
   (m) Now, there is at least a candidate for the full secret key.
   (n) For each surviving key candidate, test it with one full eight-round encryption each. If it passes the test, output it as the real key candidate, else output $\perp$.

E.5.1.3 Complexities

The trail has in total 16 bytes that must be inactive, which is approximated by a probability of 2^{-128} for a pair of pairs. For $s = 45$, we obtain $s^{12} \approx 2^{65.9}$ CPs, $n_s \approx 2^{128.4}$ mixture structures, and $2^{130.4}$ pairs. The probability that fewer than one pair satisfies the trail is below $e^{-1} \approx 0.368$. Taking into account all possible 32-bit values that an exchanged leftmost anti-diagonal in C_0 can have, each of the $2^{65.9}$ ciphertexts can change to at most

$2^{97.9}$ ACCs.

The attack iterates over $2^{130.4}$ pairs in the following:

- For each pair, first guess $4 \cdot 8$ bits and filters 8 bits until Step (d), with at most $2^{162.4}$ EEs and $2^{154.4}$ candidates left.
- For those, guess 2^{48} candidates, i.e., at most $2^{202.4}$ partial encryptions are needed up to Step (g).
- Obtain an 8-bit filter in Step (h), a 16-bit filter in Step (i), and a 16-bit filter in Step (j), so $2^{162.4}$ candidates remain.
- In Steps (k) and (l), we guess 16 bits each and obtain an eight-bit filter each, so $2^{178.4}$ full key candidates remain, which are then tested with one full encryption each.
- Therefore, the effort is at most $2^{202.4}$ EEs.

The attack needs memory for two hash tables for 2^{32} four-byte states, $2 \cdot 2^{65.9}$ chosen plaintext-ciphertext pairs and $2 \cdot 2^{65.9+32} = 2^{98.9}$ CCs in memory. In total, the attack needs approximately

- Data-collection phase: $2^{65.9}$ CPs and $2^{97.9}$ ACCs.
- Time: $2^{202.4}$ eight-round EEs.
- Memory: $2^{98.9}$ states.

E.5.2 Yoyo Attack on 10-round `Vistrutah`-512

We can adopt the five-round yoyo attack by [RBH17] to 10-round `Vistrutah`-512. However, we use it only as a variant of a mixture.

E.5.2.1 Attack

ΔZ_0^3 is mapped to a difference ΔW_0^3 with at most two active diagonals with probability approximately $\binom{4}{2} \cdot 2^{-64} \approx 2^{-61.4}$. For every plaintext pair that fulfills the transition from ΔZ_0^3 to ΔW_0^3, there are seven further plaintext pairs in the mixture structure from mixing the first diagonals of the individual plaintext slices. Moreover, for each of these ciphertext pairs, we can construct seven further ciphertext pairs by using a different set of four rows in Z_i^{11} to be exchanged. Thus, for every plaintext pair, one obtains a total of $8 \cdot 8$ plaintext pairs (eight from mixed plaintexts, $7 \cdot 8$ from their corresponding exchanged and redecrypted ciphertexts) that are usable for key recovery.

The attack involves the first diagonals of $K_i^1[0, 5, 10, 15]$ for $i \in \{0, 1, 2, 3\}$ and the full K_0^2, that correspond to the 28 secret key bytes SK[0, 5, 10, 15, 16..31, 32, 37, 42, 47, 48, 53, 58, 63], or 2^{224} key candidates.

The steps are as follows:

1. Choose four sets of s 32-bit random values each. From those construct plaintexts by iterating over all combinations and assigning them to the first diagonal of all plaintext slices to form s^4 plaintexts, leaving all other plaintext bytes random but the same for all plaintexts. Group the plaintexts into structures of 8 plaintext-diagonal mixture pairs each.
2. For all plaintexts, ask the oracle for their corresponding ciphertexts and invert their final `MixColumns` operation.
3. For all ciphertext pairs (C, C') whose plaintexts have 16 active bytes over the four first diagonals:
   (a) Exchange the values of the topmost row in some state slices to construct $C^{(2)}$ and $C^{(3)}$ from C and C' (here, we slightly abuse the notation as we really mean the states before the final `MixColumns` operation).

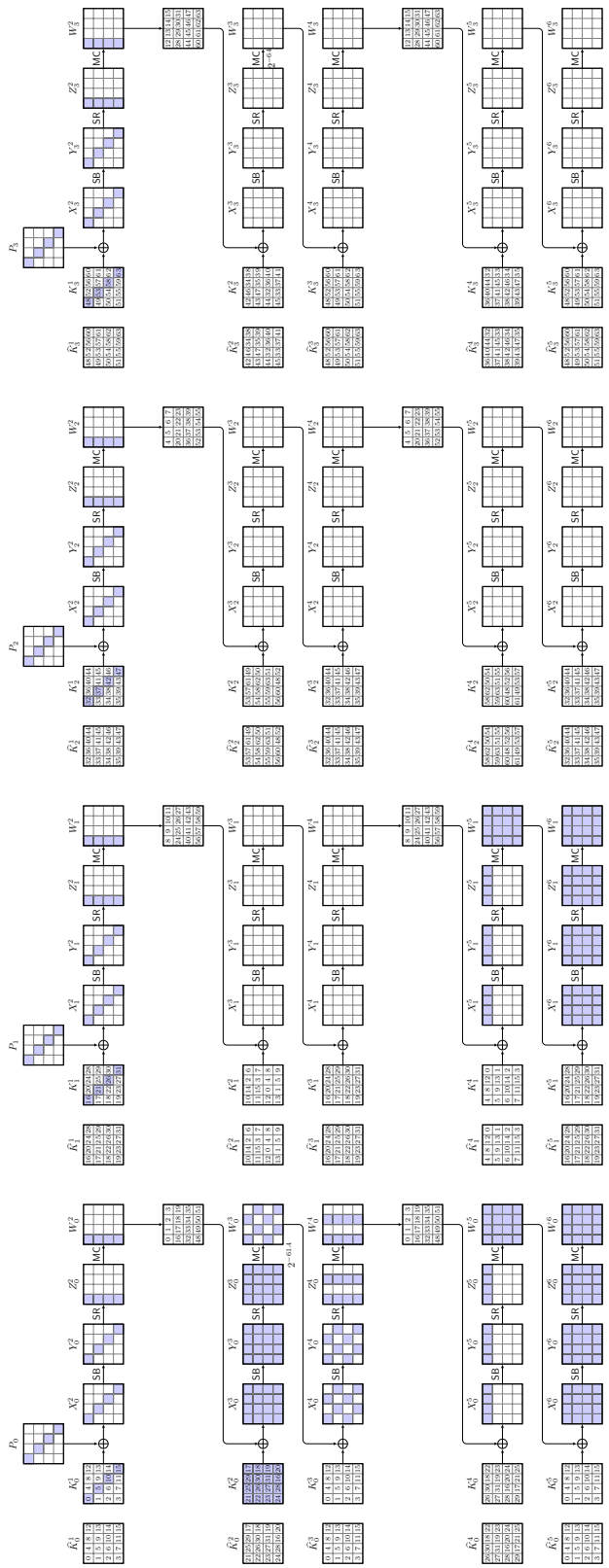


Figure 44: Yoyo attack on 10-round Vistrutah-512: Upper forward differential trail.

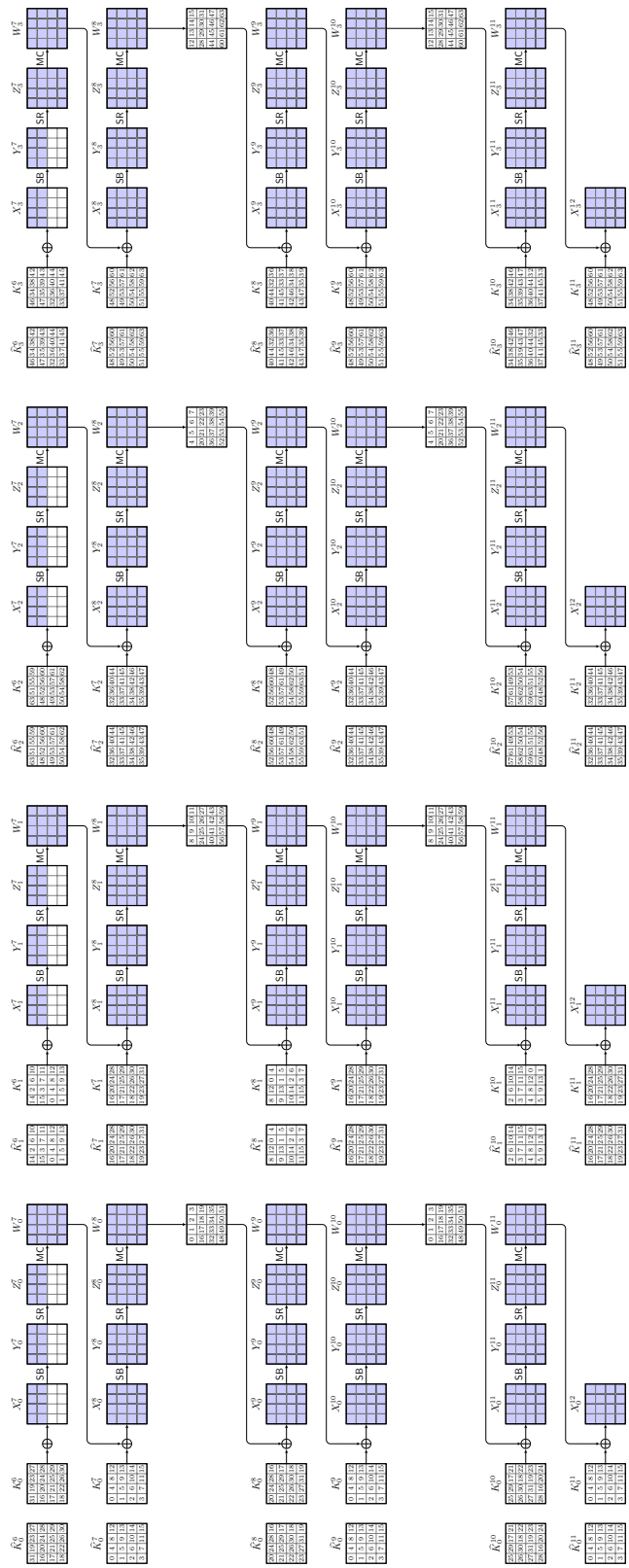


Figure 45: Yoyo attack on 10-round `Vistrutah`-512: Lower forward differential trail.

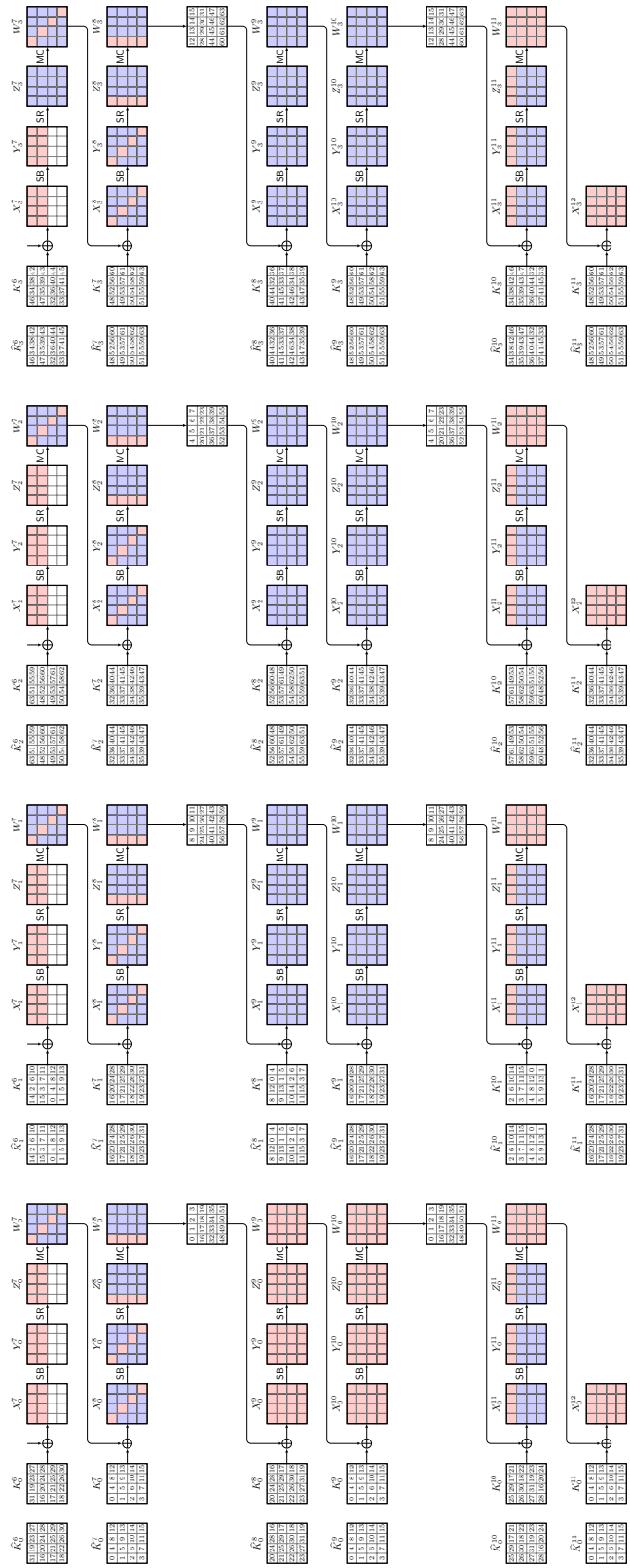


Figure 46: Yoyo attack on 10-round Vistrutah-512: Lower backward differential trail.

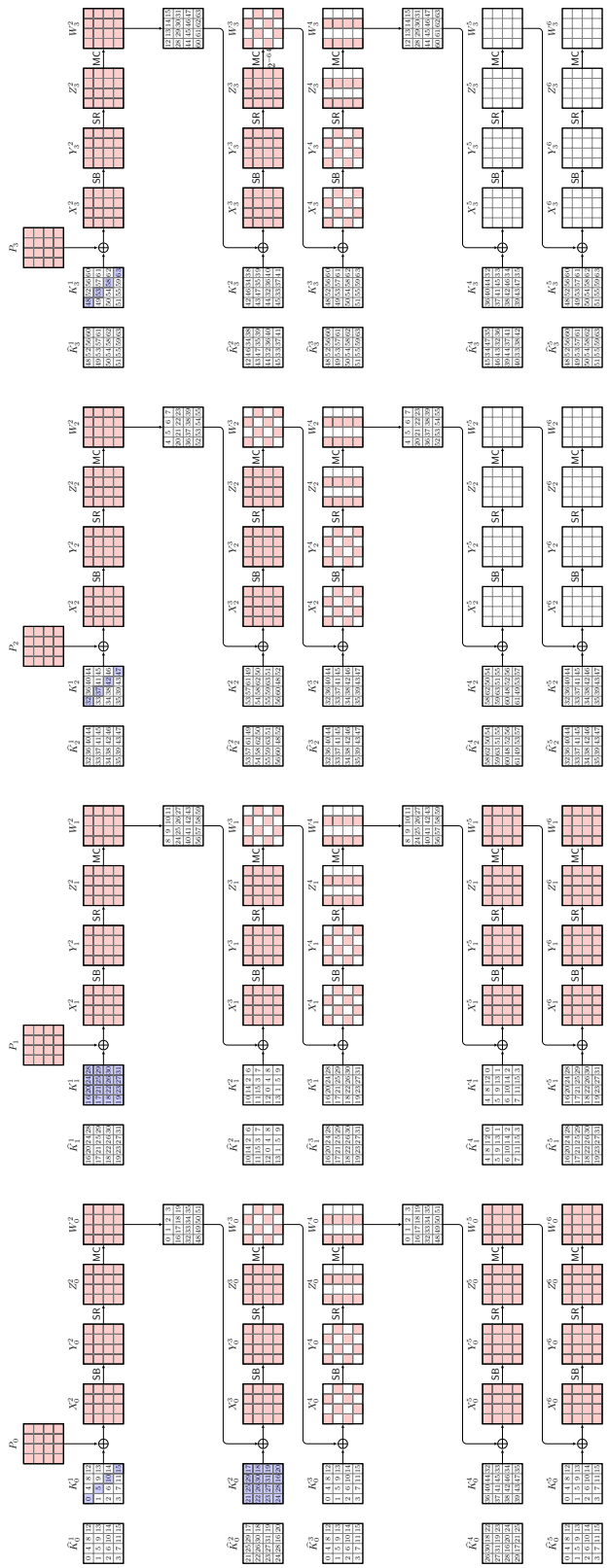


Figure 47: Yoyo attack on 10-round `Vistrutah`-512: Upper backward differential trail.

(b) Repeat the exchange process with exchanging other sets of rows between C and C'. in total, we can form seven new pairs in this way.

(c) Compute the final `MixColumns` operation and ask the oracle for the corresponding decryptions of the so-constructed mixture ciphertext pairs. The backward trails are depicted in Figure 46 and 47.

(d) In the following, we describe only the case that the active diagonals are $\mathcal{D}_{0,2}$ but the other cases work analogously. For all six versions of two active diagonals in ΔW_0^3 and for each mixture structure containing eight plaintext pairs and the corresponding $7 \cdot 8$ decrypted plaintext pairs:

i. Guess four bytes $K_1^1[0, 5, 10, 15] = SK[16, 21, 26, 31]$ and compute from $P_1[0, 5, 10, 15]$ and $P_1'[0, 5, 10, 15]$ to $\Delta X_0^3[1, 5, 9, 13]$. Since we have $K_0^2[0, 5, 10, 11]$, compute further to $X_0^3[5]$ and to $Z_0^3[1]$ and $\Delta Z_0^3[1]$ for all pairs.

ii. Guess four bytes $K_0^1[0, 5, 10, 15] = SK[0, 5, 10, 15]$ and compute from $P_0[0, 5, 10, 15]$ and $P_0'[0, 5, 10, 15]$ to $\Delta X_0^3[0, 4, 8, 12]$. With the help of $K_0^2[0]$, compute further to $X_0^3[0]$ and to $Z_0^3[0]$ and $\Delta Z_0^3[0]$ for all pairs.

iii. From $\Delta Z_0^3[0, 1]$ and under the assumption that the current structure was formed by a good pair, which implies that $\Delta W^3[1, 3] = (0, 0)$, derive $\Delta Z_0^3[2, 3] = \Delta Y_0^3[10, 15]$ and $\Delta W^3[0, 2]$ for each pair.

iv. Guess four bytes $K_2^1[0, 5, 10, 15] = SK[32, 37, 42, 47]$ and compute from $P_2[0, 5, 10, 15]$ and $P_2'[0, 5, 10, 15]$ to $\Delta X_0^3[2, 6, 10, 14]$. With the help of $K_0^2[10]$, compute further to $X_0^3[10]$ and to $Y_0^3[10]$ and ${Y'}_0^3[10]$ for all pairs. Their difference must match $\Delta Y_0^3[10]$ derived from the previous step for each pair, which yields a one-byte filter per pair. While the input pairs are dependent and an independence assumption obviously does *not* hold, the resulting $7 \cdot 8$ pairs result from decryption by the oracle. Here, we argue that we can use an independence assumption in the plaintexts that is the states (in decryption direction) two rounds after the eight-round yoyo distinguisher. As a result, we argue that we have a considerable filter of up to 56 bytes on average. Thus, we can discard all structures that do not satisfy this filter.

v. For the fraction of structures which survive, guess four bytes $K_3^1[0, 5, 10, 15] = SK[48, 53, 58, 63]$ and compute from $P_3[0, 5, 10, 15]$ and $P_3'[0, 5, 10, 15]$ to $\Delta X_0^3[3, 7, 11, 15]$. With the help of $K_0^2[11]$, compute further to $X_0^3[11]$ and to $Z_0^3[15]$ and $\Delta Z_0^3[15]$ for all pairs. At this stage, the full ΔX^3 is known for all pairs.

vi. Guess $\Delta W_0^3[5, 7, 8, 10, 13, 15]$ for the first pair, so that the full ΔW_0^3 is known for the first pair. Derive from it its full ΔY_0^3. With the help of the full ΔX_0^3 and the property that there exists on average one value for an input-output differential of the S-box, derive the full states X_0^3 and ${X'}_0^3$ for the first pair and therefore, derive the full K_0^2.

vii. For all other pairs, verify if the current guess of K^2 is correct by encrypting them all to ΔW_0^3 and check for having only the desired two diagonals active. If not, discard the current guess. Otherwise, output the 28-byte key candidate that survives the filter for all pairs.

(e) Repeat the procedure for all surviving structures, using only the pairs that resulted from decryption and K_1^2 and ΔW_1^3 to obtain 20 more bytes of the key, i. e. $SK[1..4, 6..9, 11..14, 34, 39, 40, 45, 50, 55, 56, 61]$.

(f) Given then 48 bytes of the secret key, guess the remaining 16 bytes and test each candidate with one full encryption each.

E.5.2.2 Complexities

For a given s, s^4 CPs are constructed that can be clustered into $(s \cdot (s-1)/2)^4$ mixture structures. Since the probability for one pair is approximately $2^{-61.4}$, taking $s = 310$, we can form $2^{33.1}$ CPs that can be combined to $2^{62.2}$ mixture structures of eight pairs each. On the ciphertext end, we obtain $2^{65.2}$ ciphertext pairs among which a fraction of approximately $(255/256)^{16} \approx 0.94$ on average differ in all 16 bytes in the first diagonal of all state slices, corresponding to $2^{62.1}$ mixture structures and $2^{65.1}$ pairs on average. Each of those can be mixed in seven ways, which yields $2^{67.9}$ adaptively chosen ciphertext pairs that are re-encrypted. Thus, the time complexities of the data collection consist of approximately $2^{33.1}$ encryptions, $2^{68.9}$ decryptions, and $2^{68.9}$ MAs to store them.

The key-recovery phase consists of guessing twelve bytes for each structure of 64 plaintext pairs, and computing $12 + 4$ S-boxes for each guess out of $10 \cdot 64$ S-boxes in the full cipher, i.e., $2^{62.1} \cdot 64 \cdot 2 \cdot 2^{8 \cdot 12} \cdot 16/640 \approx 2^{159.8}$ EEs before the first filter. After that, we expect on average two correct mixture structure(s) to remain. For each of them, we guess four more key bytes and six bytes from ΔW_0^3 that yield the full K_0^2. We can filter all but the correct values with the help of all pairs with at most $2 \cdot 64 \cdot 2^{10 \cdot 8} \cdot (4 + 12)/640 \approx 2^{81.7}$ EEs. Thus, the effort is upper bounded by $2^{159.8}$ EEs. We expect a similar effort for the second stage to recover 20 more key bytes, plus 2^{128} encryptions to find the remaining 16 key bytes on average.

The attack needs memory for a hash table for $2^{33.1} + 2^{68.9} \approx 2^{68.9}$ states. In total, the attack needs approximately:

- Data-collection phase: $2^{33.1}$ CPs and $2^{68.9}$ ACCs.
- Time: $2^{159.8}$ eight-round EEs.
- Memory: $2^{68.9}$ states.

E.6 Multiple-of-n Cryptanalysis

E.6.1 Multiple-of-2 Distinguisher on 7-round `Vistrutah`-256

We can easily find a Multiple-of-2 distinguisher on seven-round `Vistrutah`-256 as shown in Figure 48.

E.6.1.1 Distinguisher

We choose plaintexts iterating over all 2^{128} values of P_0 and leaving P_1 constant. Those can be split into two subsets at W_0^2, where the even-indexed bytes are mapped to X_0^3 and the odd-indexed bytes to X_1^3, as colored in blue and red, respectively, in the figure. The different blue and red partitions affect each other only in ΔW_0^5 and ΔW_1^5. For any such pair that produces any inactive diagonal at ΔW_i^5, there exists a mixture pair at ΔW_0^2 by swapping either the blue or the red bytes. Thus, whenever a pair exists, there exist at least two and a multiple of that number as pairs. The inactive diagonal at ΔW_i^5 can be detected up to ΔZ^7, which yields a seven-round distinguisher with 2^{128} CPs. Moreover, one can easily cut off the first round and obtain a six-round distinguisher with complexity 2^{32} CPs by iterating over any plaintext diagonal and leaving all other plaintext bytes constant. The distinguisher will detect the real round-reduced `Vistrutah`-256 with probability one and falsely declares a random permutation as real with a probability at most 1/2.

- Computations: 2^{128} EEs and 2^{132} MAs to huge tables.
- Data: At most 2^{128} CPs.
- Memory: 2^{129} states for the plaintext-ciphertext pairs.

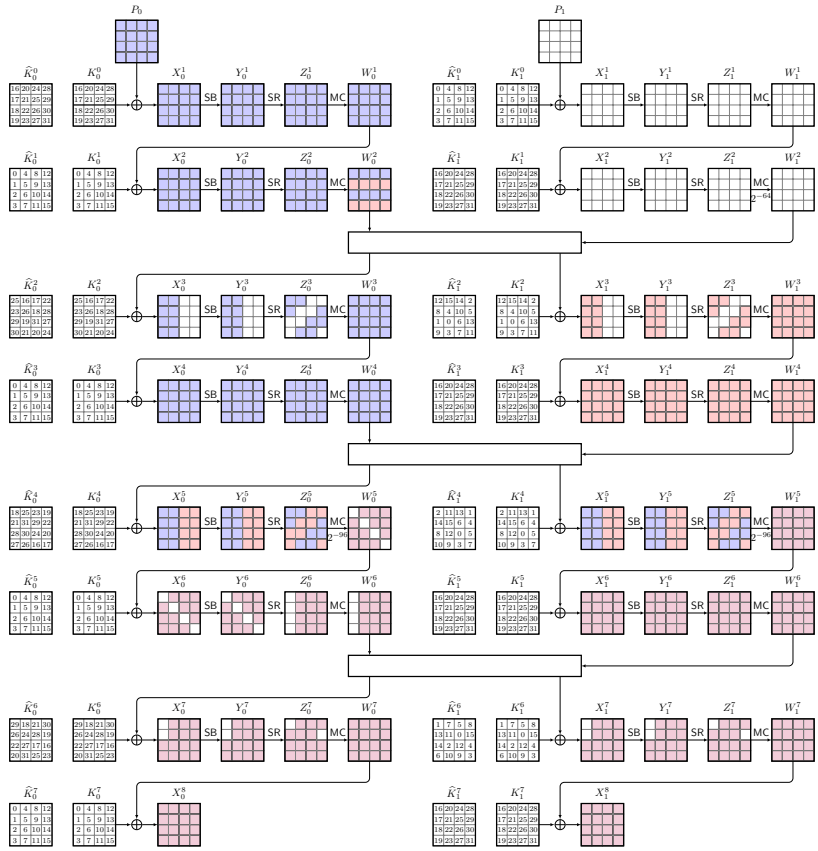


Figure 48: Multiple-of-2 distinguisher on seven-round Vistrutah-256.

E.6.2 Multiple-of-8 Distinguisher on 10-round Vistrutah-512

The Multiple-of-8 distinguisher on five-round AES from [GRR17] applies similarly to five steps of Vistrutah-512, here from Rounds 2–11.

E.6.2.1 Distinguisher

An exemplary distinguisher is shown for Rounds 2–6 in Figure 49 and for Rounds 7–11 in Figure 50. We choose plaintexts from the set of the first diagonals in every plaintext's slices $P_0, \ldots, P_3$. Those texts iterate over all values of the diagonals of W_0^3, which are mapped independently from each other to the state slices of W^6 and the rows of Z^7. Let $(d_0^0, d_1^0, d_2^0, d_3^0)$ and $(d_0^1, d_1^1, d_2^1, d_3^1)$ denote the diagonals of W_0^3 and ${W'}_0^3$ for a pair of texts. Whenever it satisfies to have the j-th diagonal in all slices ΔW_i^7 for all $i \in \{0, 1, 2, 3\}$ inactive, the same holds for any other plaintext pair that has an exchanged version of those four diagonals at W_0^3 and ${W'}_0^3$, e.g. $(d_0^1, d_1^0, d_2^0, d_3^0)$ and $(d_0^0, d_1^1, d_2^1, d_3^1)$. Since there are eight (unordered) pairwise distinct exchanged pairs in a plaintext structure, whenever such a pair exists, there exist a multiple of eight such pairs.

We use the distinguisher as follows: We ask for the encryption of 2^{128} CPs, count all potential pairs with at least one inactive row in all state slices ΔZ^{11}, and output real if that number is a multiple of eight and random otherwise. The real 10-round Vistrutah-512 will be detected with probability one and a random permutation will be wrongly detected as real with a probability $1/8$.

- Computations: 2^{128} EEs and 2^{131} MAs to huge tables.

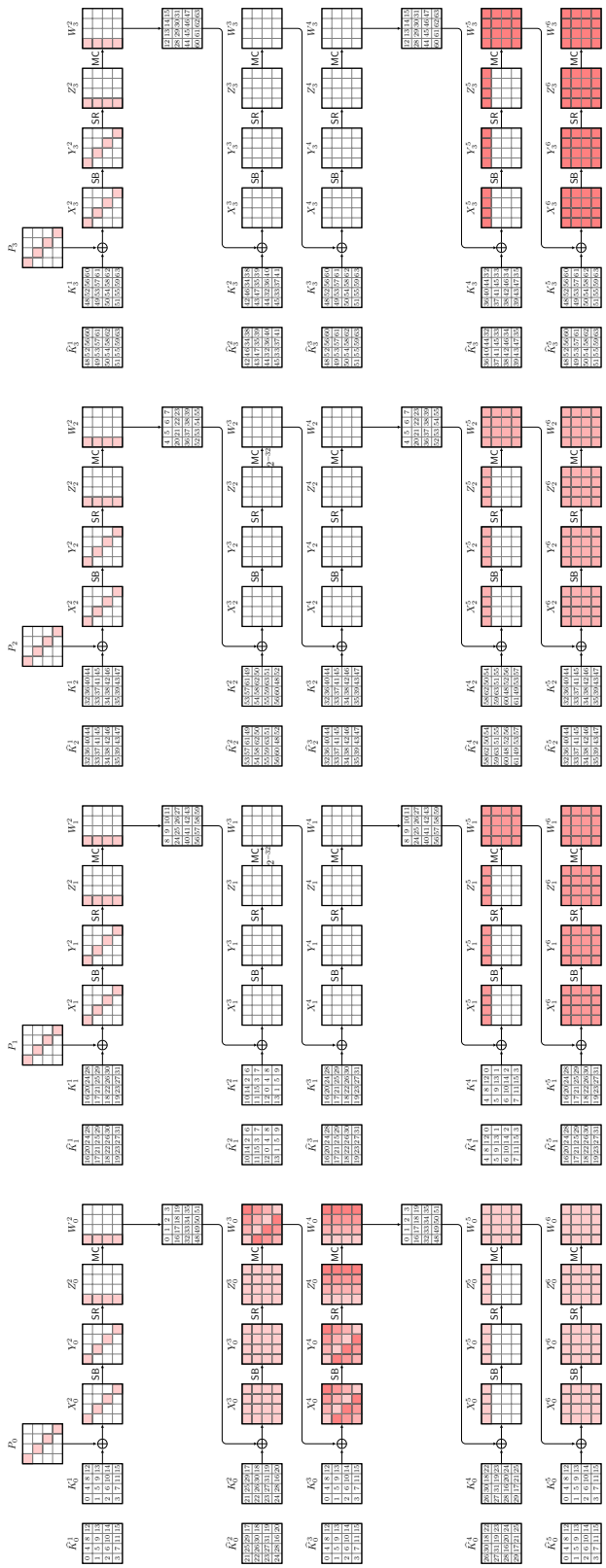


Figure 49: Multiple-of-8 distinguisher on 10-round `Vistrutah`-512: Rounds 2–6.

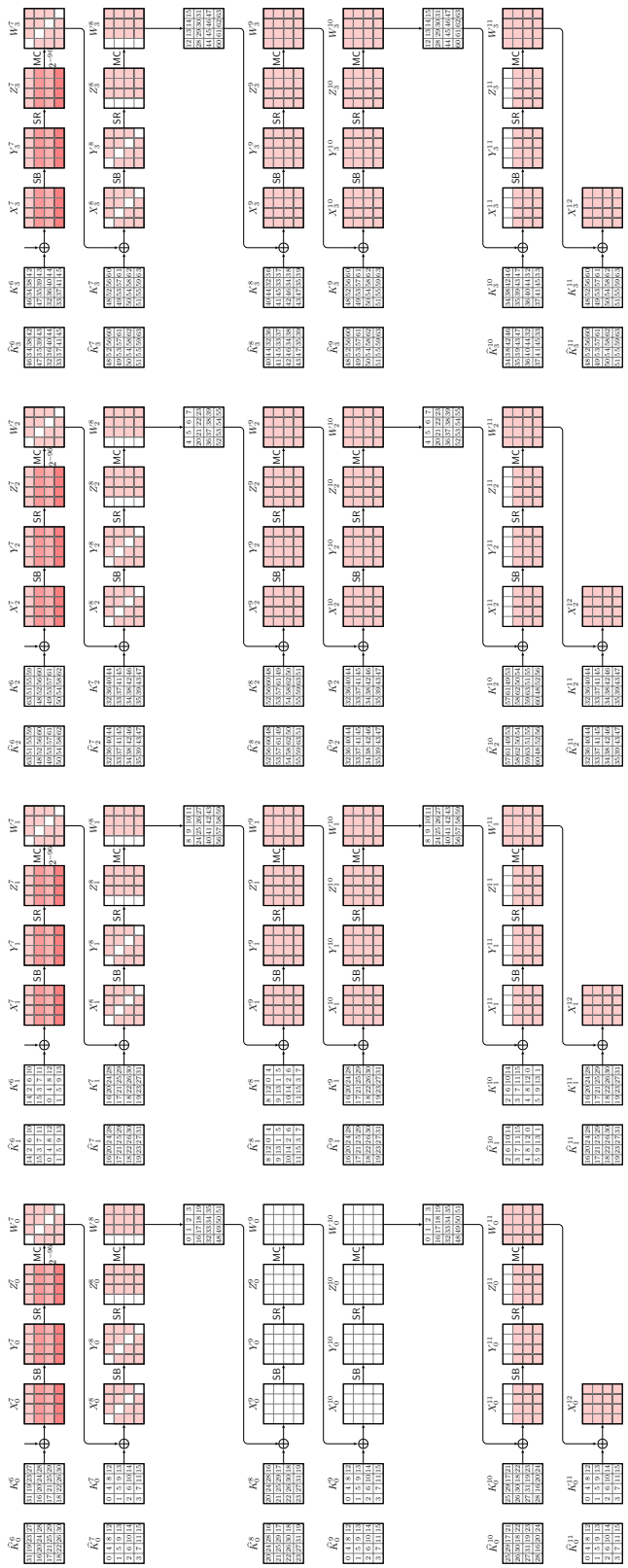


Figure 50: Multiple-of-8 distinguisher on 10-round Vistrutah-512: Rounds 7–11.

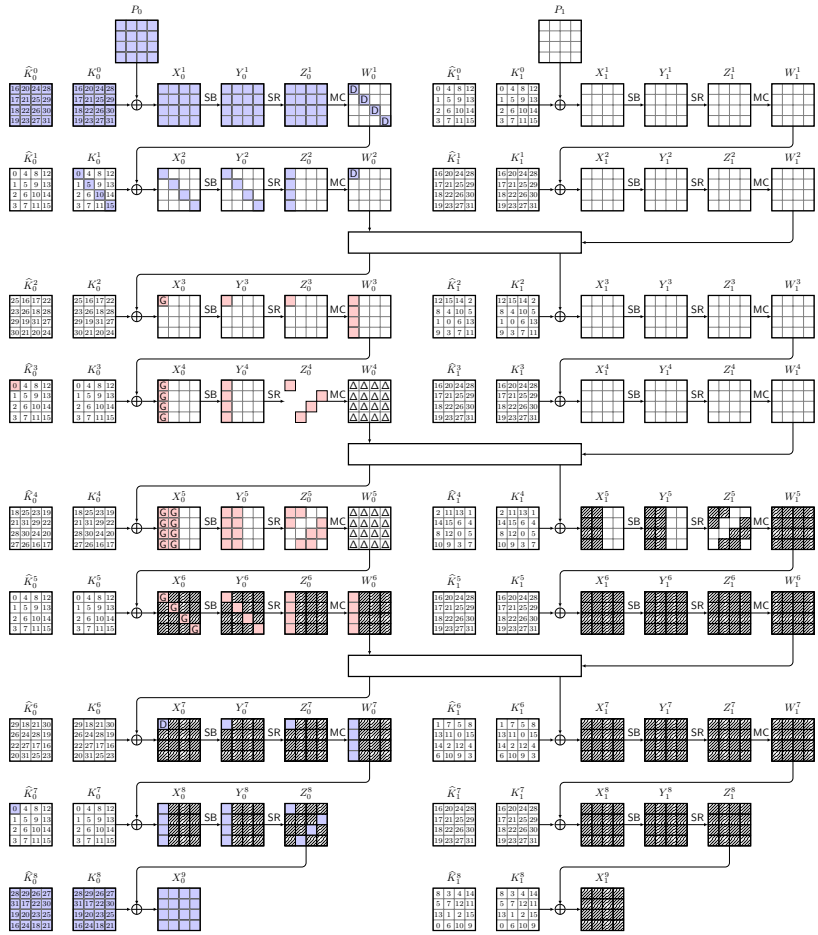


Figure 51: 8-round DS-MitM attack on `Vistrutah`-256: Distinguisher and key-recovery phases.

- Data: At most 2^{128} CPs.
- Memory: 2^{130} states.

E.7 Demirci-Selçuk Meet-in-the-middle Cryptanalysis

E.7.1 DS-MitM Attack on 8-round `Vistrutah`-256

E.7.1.1 Offline Phase

The trail in the offline phase is given in red in Figure 51, covering Rounds 3 through 6.

We take a set of a text at X_0^3 and s 30 related texts $X_0^{3,i}[0] = X_0^3[0] \oplus \langle i \rangle$ and all other bytes being constant. We guess in total 17 bytes in the offline trail from $X_0^3[0]$ to $W_0^6[0]$:

- $X_0^3[0]$,
- $X_0^4[0..3]$,
- $X_0^5[0..7]$, and
- $X_0^6[0, 5, 10, 15]$.

These bytes allow the computation of $W_0^{6,i}[0]$ for all related texts. We also obtain one key byte $\widehat{K}_0^3[0]$ that is not relevant in the rest. We then compute the lead pair and 30

related texts through the 4-round distinguisher, which only has 20 active S-boxes that have to be computed, compared to $8 \cdot 32 = 256$ in the eight-round cipher — and store 32 byte differences to the lead pair as an entry into a table $\mathcal{L}$. The complexity is about $2^{17 \cdot 8} \cdot 32 \cdot 20/256 \approx 2^{137.3}$ EEs and $2^{17 \cdot 8} = 2^{136}$ MAs to $\mathcal{L}$. The memory complexity is $2^{17 \cdot 8} \cdot 32/32 \approx 2^{136}$ states.

E.7.1.2 Online Phase

In the online phase, we construct 2^{61} random but distinct 128-bit plaintext values for P_0 that can form approximately 2^{121} plaintext pairs. The plaintext slice P_1 is chosen independently at random but identical for all plaintexts. We ask the oracle for all corresponding ciphertexts and stores all resulting texts of the current structure in a table. For all pairs of plaintext-ciphertext pairs $((P, C), (P', C'))$:

1. Assume that the pair follows the differential in Figure 51;
2. Gess $\Delta W_0^1[0, 5, 10, 15]$ and $\Delta W_0^2[0]$ and derive the values in the first two rounds, K_0^0, and $K_0^1[0, 5, 10, 15]$. This yields $SK[0, 5, 10, 15, 16..31]$.
3. From the plaintext whose corresponding integer representation of P_0 is lower, derive 30 related texts at $W_0^2[0]$, partially decrypt to their plaintexts, and ask for their corresponding ciphertexts.
4. Partially decrypt all of them to $\widehat{W}_0^7 = \texttt{MixColumns}^{-1}(X_0^8)$.
5. For C and C', guess $\Delta X_0^7[0]$ and derive $\widehat{K}_0^7[0]$.
6. Decrypt all texts to $X_0^7[0]$ and derive the sequence of differences $X_0^7[0]$, ${X'}_0^7[0]$, $X_0^{7,1}[0]$, ..., $X_0^{7,30}[0]$ and look it up in $\mathcal{L}$.
7. If a match is found, guess the remaining 11 key bytes in $SK[0..15]$ and test the correctness of the key candidate with one encryption.
8. If the test succeeds, output the key candidate.

The online phase needs at most $2^{61} + 2^{121} \cdot 32 \approx 2^{126}$ CPs that have to be encrypted. We then guess six bytes in the key-recovery phase for each pair. The first two rounds have 20; the last two rounds 5 active S-boxes that have to be computed for each of 32 texts involved for each pair. Since $\mathcal{L}$ has $2^{17 \cdot 8}$ 32-byte entries, and the online phase looks up $\mathcal{L}$ $2^{121+6 \cdot 8} = 2^{169}$ times, we expect $2^{136+169-256} = 2^{49}$ key candidates on average. Each of those require 11 remaining bytes to be guessed and tested with on average 2^{137} encryptions. The average complexity therefore is approximately $2^{61} + 2^{121} \cdot 32 + 2^{121} \cdot 2^{6 \cdot 8} \cdot 32 \cdot 25/256 + 2^{137} \approx 2^{170.6}$ EEs, plus $2 \cdot 2^{121}$ MAs into the ciphertexts table and 2^{169} MAs into $\mathcal{L}$. The online phase has to store 2^{61} plaintext-ciphertext pairs. In total, the attack needs approximately

- Data-collection phase: 2^{126} EEs, 2^{122} MAs, 2^{126} CPs, 2^{62} states of memory.
- Offline phase: $2^{137.1}$ EEs, 2^{136} MAs, 2^{136} states of memory.
- Online phase: $2^{170.6}$ EEs and 2^{169} MAs.

E.7.2 DS-MitM Attack on 8-round Vistrutah-512

E.7.2.1 Offline Phase

The 4-round distinguisher used in the offline phase is given in red in Figure 52, covering Rounds 3 through 6. We take a set of a text at X_0^3 and $s = 36$ related texts $X_0^{3,i}[0] = X_0^3[0] \oplus \langle i \rangle$ and all other bytes being constant. We guess in total 13 bytes in the offline trail from $X_0^3[0]$ to $W_0^6[0]$:

- $X_0^3[0]$,
- $X_0^4[0..3]$,

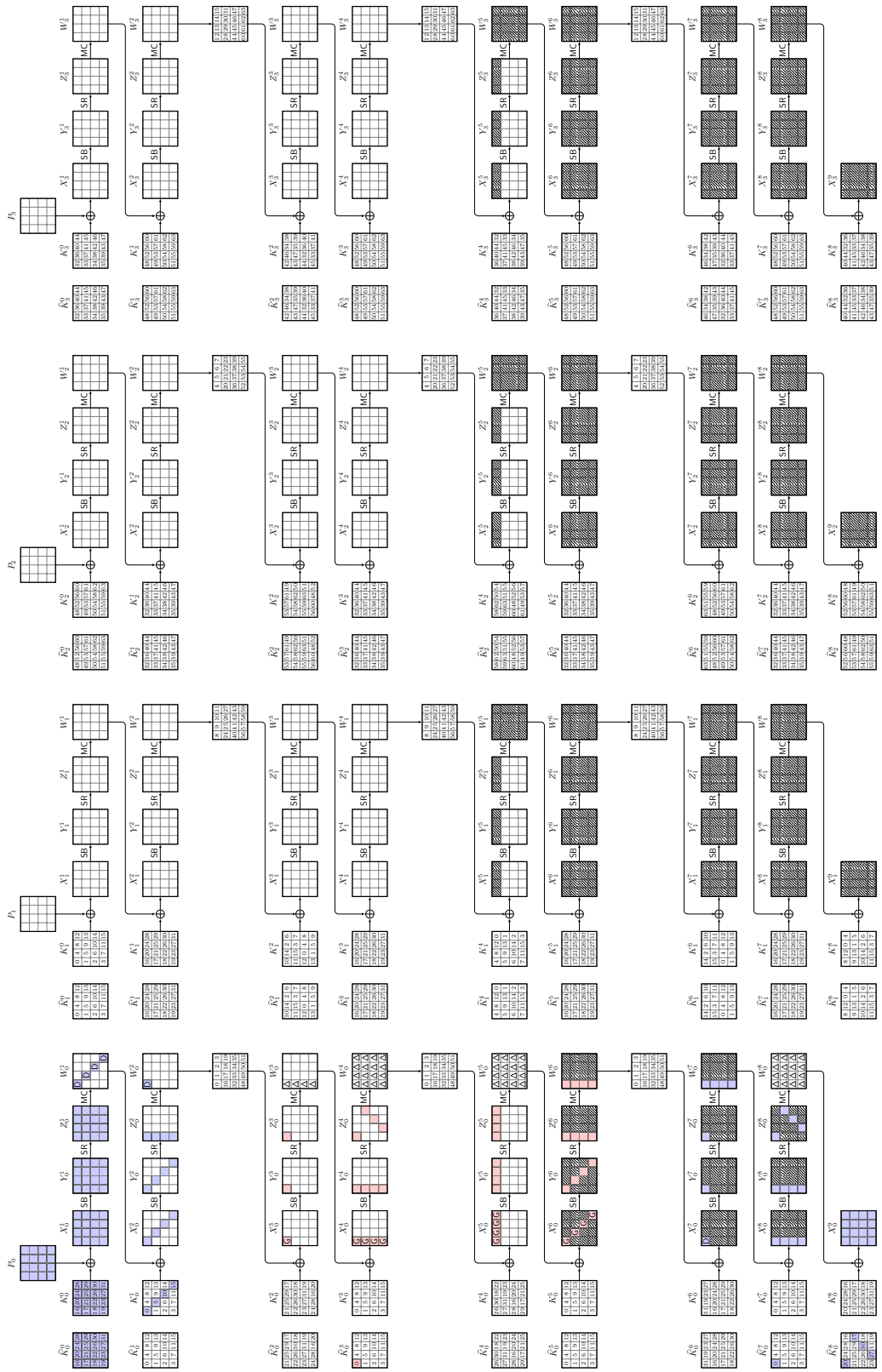


Figure 52: 8-round DS-MitM attack on `Vistrutah`-512: Distinguisher and key-recovery phases.

- $X_0^5[0,4,8,12]$, and
- $X_0^6[0,5,10,15]$.

Those allow us to compute $W_0^{6,i}[0]$ for all related texts. We also obtain one key byte $\widehat{K}_0^3[0]$ which is not further used. We compute the lead text and 36 related texts through the distinguisher, which has 13 active S-boxes that have to be computed compared to $8{\cdot}64 = 512$ in the eight-round cipher. We store 36 single-byte differences $W_0^6[0] \oplus W_0^{6,i}[0]$ to the lead text as an entry into a table $\mathcal{L}$. The complexity is about $2^{13\cdot 8} \cdot 37 \cdot 13/512 \approx 2^{103.9}$ EEs and $2^{13\cdot 8} = 2^{104}$ MAs to $\mathcal{L}$. Approximating the MAs by EEs, we get $2^{103.9} + 2^{104} \approx 2^{105.0}$ EEs. The memory complexity is $2^{13\cdot 8} \cdot 36/64 \approx 2^{103.2}$ states.

E.7.2.2 Online Phase

In the online phase of the attack, we construct 2^{61} random but distinct 128-bit plaintext values for P_0 that can form approximately 2^{121} plaintext pairs. The plaintext slices P_1, P_2, and P_3 are chosen independently at random but identical for all plaintexts. We ask the oracle for all corresponding ciphertexts and store all resulting texts of the current structure in a table. For all approximately 2^{121} pairs of plaintext-ciphertext pairs $((P,C),(P',C'))$, we conduct the following steps:

1. We assume that the pair follows the differential in Figure 52. We guess $\Delta W_0^1[0,5,10,15]$ and $\Delta W_0^2[0]$ and derive the values in the first two rounds, K_0^0, and $K_0^1[0,5,10,15]$. They yield $SK[0,5,10,15,16..31]$.
2. From the plaintext from the pair whose integer representation of P_0 is lowest, let us call it P, we derive 31 related texts $W_0^{2,i}[0]$ as the only byte that each of them differs from W_0^2, partially decrypts them to their corresponding plaintexts, and ask for their corresponding ciphertexts.
3. We partially decrypt all of them to $\widehat{W}_0^8 = \texttt{MixColumns}^{-1}(X_0^9)$.
4. Using $\widehat{K}_0^8[0,7,10,13]$ that is known from $SK[16..31]$, we guess $\widehat{K}_0^7[0]$ and decrypt all texts to $X_0^7[0]$ to obtain the sequence of differences $X_0^7[0] \oplus X_0^{7,1}[0]$, …, $X_0^7[0] \oplus X_0^{7,36}[0])$ and look it up in $\mathcal{L}$.
5. The key candidates that produce a match are returned.

E.7.2.3 Online Complexity

The online phase needs at most $2^{61} + 2^{121} \cdot 2^{40} \cdot 36 \approx 2^{166.2}$ CPs. The online complexity consists of the following steps:

- Encrypt 2^{61} initial CPs, then store them into a list $\mathcal{C}$ and read them from it with 2^{62} MAs;
- Compute 20 active S-boxes in the first two rounds for all 2^{121} pairs, five byte guesses, and 38 texts, which yields $2^{121} \cdot 2^{40} \cdot 38 \cdot 20/512 \approx 2^{161.6}$ encryptions;
- Encrypt $2^{166.2}$ derived plaintexts;
- For all pairs and guesses of K^0 and K^1, guess another eight bits of $\widehat{K}_0^7[0]$ and compute five active S-boxes for 37 texts, which needs $2^{121+40+8} \cdot 37 \cdot 5/512 \approx 2^{167.5}$ EEs; and
- 2^{169} MAs to $\mathcal{L}$.

Since $\mathcal{L}$ has $2^{13\cdot 8}$ 36-byte entries, and the online phase looks up $\mathcal{L}$ $2^{121+6\cdot 8} = 2^{169}$ times, we expect $2^{104+169-256} = 2^{-15}$, i.e., only a few false positive and the correct key candidates to survive on average; The online phase can be repeated three further times, reusing the same table $\mathcal{L}$ but shifting the plaintext-ciphertext trails from different values in P_0 to different values in P_1, P_2, or P_3 and consider C_1, C_2, or C_3 at the ciphertext side accordingly. Only a few encryptions are needed to verify the few full key candidates remaining at the end.

The average complexity therefore is approximately $4 \cdot (2^{61} + 2^{161.6} + 2^{166.2} + 2^{167.5}) \approx 2^{170.0}$ EEs plus $4 \cdot (2^{62} + 2^{169}) \approx 2^{171}$ MAs. If we approximate them by a full encryption,

we obtain $2^{170} + 2^{171} \approx 2^{171.6}$ EEs. The online phase has to store 2^{61} plaintext-ciphertext pairs at a time.

E.7.2.4 Total Complexity

In total, the attack needs approximately

- Data: $2^{166.2}$ CPs.
- Offline phase: $2^{103.9}$ EEs, 2^{104} MAs, $2^{103.2}$ states of memory.
- Online phase: $2^{170.0}$ EEs and 2^{171} MAs.

E.7.3 DS-MitM Attack on 10-round `Vistrutah`-256

E.7.3.1 Offline Phase

We fix in total 2^c in- and output differences $\Delta X_0^3[0]$ and $\Delta W_0^8[0,2]$. The probability of a pair to follow the differential trail is $2^c \cdot 2^{-128} \cdot 2^{-256} = 2^{c-384}$. For $c = 4$, we have a probability of 2^{-380}. Then, the offline phase must guess a total of 36 bytes:

- $\Delta Z_0^3[0]$,
- $X_0^4[0..3]$,
- $X_0^5[0..7]$ and $X_1^5[0..7]$,
- $\Delta X_0^7[0,2,3,4,5,7,8,9,10,12,13,14]$, and
- $\Delta X_0^8[0,5,10]$.

Those allow us to compute $\Delta W_0^8[0] \oplus \Delta W_0^8[2] = \Delta X_0^9[0] \oplus \Delta X_0^9[1]$ for all related texts. We show that due to the key-schedule relations, there are at least five byte constraints in the offline phase that will reduce the guessing amount:

- From $\widehat{K}_0^3[0]$ and $\widehat{K}_0^5[0]$, we get an overlap in the same byte.
- From $\widehat{K}_1^5[0,1,4,7,10,11,13,14]$ and $K_0^6[0,2..5,7..10,13..15]$, we obtain 20 bytes for the secret-key bytes 16..31, which yields to a 2^{-32}-filter on average.

Moreover, we obtain $2 + 8 + 3 = 13$ byte relations about the secret-key bytes 0..15 from $\widetilde{K}_0^4[0,10]$, $\widehat{K}_0^5[0,1,4,7,10,11,13,14]$, and $K_0^7[0,5,10]$ that could be used for matching. Thus, each trail yields $16 + 13 = 29$ key bytes in total.

We compute the lead pair and 16 related texts through the 6-round distinguisher, which has 68 active S-boxes that have to be computed compared to $10 \cdot 32 = 320$ in the 10-round cipher. We stores 29 key bytes and 16 byte differences to the lead pair, i.e., 45 bytes in total, as an entry into a table $\mathcal{L}$. For $c = 4$, the complexity is approximately $2^4 \cdot 2^{31 \cdot 8} \cdot (2 + 2^4) \cdot 68/320 \approx 2^{253.9}$ EEs and $2^{4+248} = 2^{252}$ MAs to $\mathcal{L}$. The memory complexity is $2^4 \cdot 2^{31 \cdot 8} \cdot 45/32 \approx 2^{252.5}$ states.

E.7.3.2 Online Phase

In the online phase, we construct 2^s structures of 2^{128} plaintexts each, iterating over all values in P_0 where each structure can form 2^{255} pairs. To have 2^{380} pairs, we set $s = 125$, which yields a data complexity of 2^{253} CPs. We ask the oracle for all corresponding ciphertexts, inverts the final `MixColumns` operation, and stores all resulting texts of the current structure in a table. It identifies pairs with inactive inverse diagonals $\mathcal{ID}_{2,3}(X_0^{11})$ and $\mathcal{ID}_{0,3}(Z_0^{10})$, which has probability 2^{-192}. Thus, $2^{380-192} = 2^{188}$ pairs survive on average.

For each surviving pair, we have to guess at most eight key bytes:

- $\Delta X_0^2[0,5,10,15]$,
- $\Delta Z_0^9[0,13]$, and
- $Y_0^9[0,1]$.

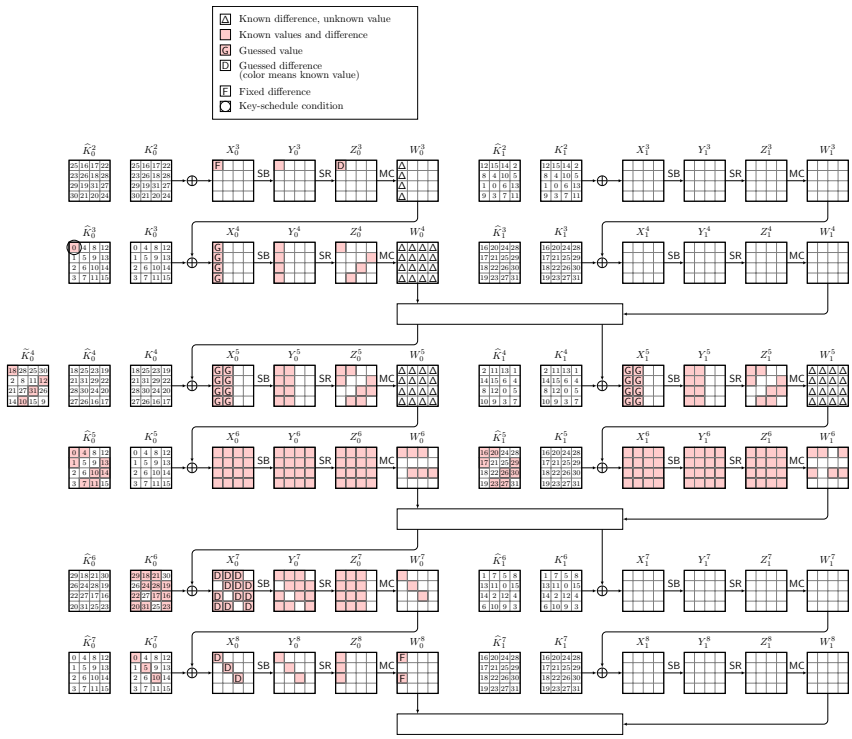


Figure 53: The offline distinguisher in the attack on Vistrutah-256.

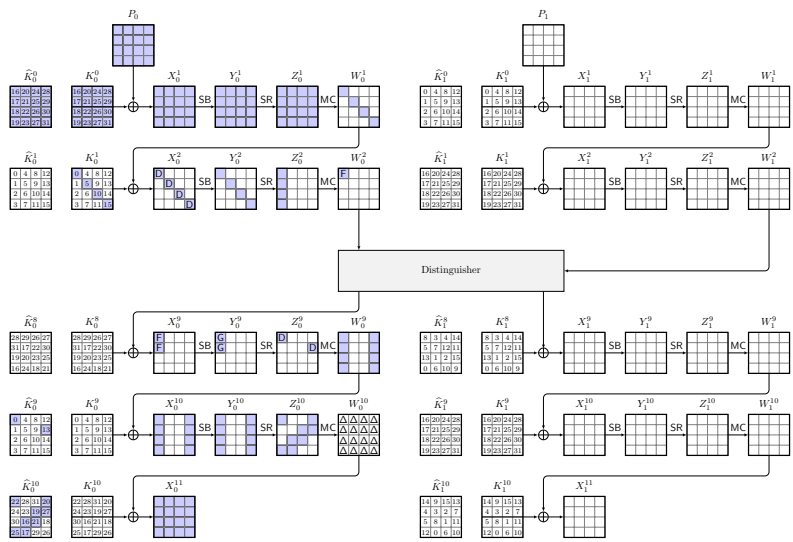


Figure 54: The outer rounds of the online filtering and key-recovery phase for the 10-round attack on Vistrutah-256.

However, we obtain the full secret-key bytes 16..31 from the top phase that yield eight constraints. The trail has at most 20 active S-boxes in the first two rounds and 10 active S-boxes in Rounds 9 and 10. First, we guess the four bytes in X_0^2, and obtain their plaintexts and the full left initial round key. This gives us also the final left round key, and allows to partially decrypt and check the ciphertext pair if their difference has only two active bytes at ΔZ_0^9. Only if this is the case, with probability 2^{-48}, we guess $\widehat{K}_0^9$ to compute to $X_0^9[0,1]$ and $X'^9_0[0,1]$, and check if it is one of the 2^c constraint 8-bit differences. If so, we build the partial δ-set of related texts from W_0^2, derive their plaintexts, look up their ciphertexts, and partially decrypt them X_0^9 and look up the key. The complexity is at most

$$\begin{aligned} &2^{188} \cdot 2^{4\cdot 8} \cdot 2 \cdot 20/320 + \\ &2^{188+4\cdot 8} \cdot 2 \cdot 8/320 + \\ &2^{188+4\cdot 8} \cdot 2^{-48} \cdot 2^{16} \cdot 2 \cdot 2/320 + \\ &2^{188+4\cdot 8} \cdot 2^{-48+16-8+c} \cdot 16 \cdot 30/320 \approx 2^{217} + 2^{215.7} + 2^{181.7} + 2^{200.6} \approx 2^{217.5} \text{ EEs} . \end{aligned}$$

Afterwards, we expect $2^{188+32-48+16-8+c} = 2^{184}$ lookups into a table of 2^{252} entries and have 32 bytes for matching, the full secret key 16..31 and 16 differences of the related texts at $\Delta X_0^9[0] \oplus X_0^9[1]$. Thus, we expect $2^{184+252-256} = 2^{180}$ candidates. For each, we can guess three remaining key bytes and perform a test encryption with at most 2^{204} encryptions in total.

E.7.3.3 Total Complexity

In total, the attack needs approximately

- Data-collection phase: 2^{253} EEs, 2^{254} MAs, 2^{253} CPs, 2^{128} states of memory.
- Offline phase: $2^{253.9}$ EEs, 2^{252} MAs, $2^{252.5}$ states of memory.
- Online phase: $2^{217.5}$ EEs and 2^{184} MAs.

E.7.4 DS-MitM Attack on 12-round `Vistrutah`-512

We can mount a DS-MitM attack on 12 rounds, based on an offline distinguisher that covers $3 + 3$ rounds, which is then extended by three rounds at both ends in an online key-recovery phase, covering Rounds 2–13.

E.7.4.1 Offline Phase

The offline phase is depicted in Figure 55, starting from a single active byte in the entire state at $\Delta X_0^5[0]$ to a single active byte in ΔW_0^{10}, here at $\Delta W_0^{10}[0]$, and only the leftmost columns active in ΔW_i^{10} for $i \in \{1,2,3\}$. We restrict the number of possible input and output differences in $\Delta X_0^5[0]$ and $\Delta W_0^{10}[0]$ to 2^c options. The probability of a pair to follow the differential trail is $2^c \cdot 2^{-128} \cdot 2^{-416} = 2^{c-544}$. For $c = 4$, we have a probability of 2^{-540}.

In the offline phase we guess a total of 41 bytes:

- $\Delta Z_0^5[0]$,
- $X_0^6[0..3]$,
- $X_i^7[0,4,8,12]$ for all $i \in \{0,1,2,3\}$,
- $X_i^8[0,5,10,15]$ for all $i \in \{0,1,2,3\}$,
- $X_0^{10}[0,5,10,15]$

Those allow to compute $\Delta W_0^{10}[0,1,2,3]$ for all related texts, which produce $\Delta X_0^{11}[0,4,8,12]$. From Figure 55, we see that the distinguisher defines 23 key bytes:

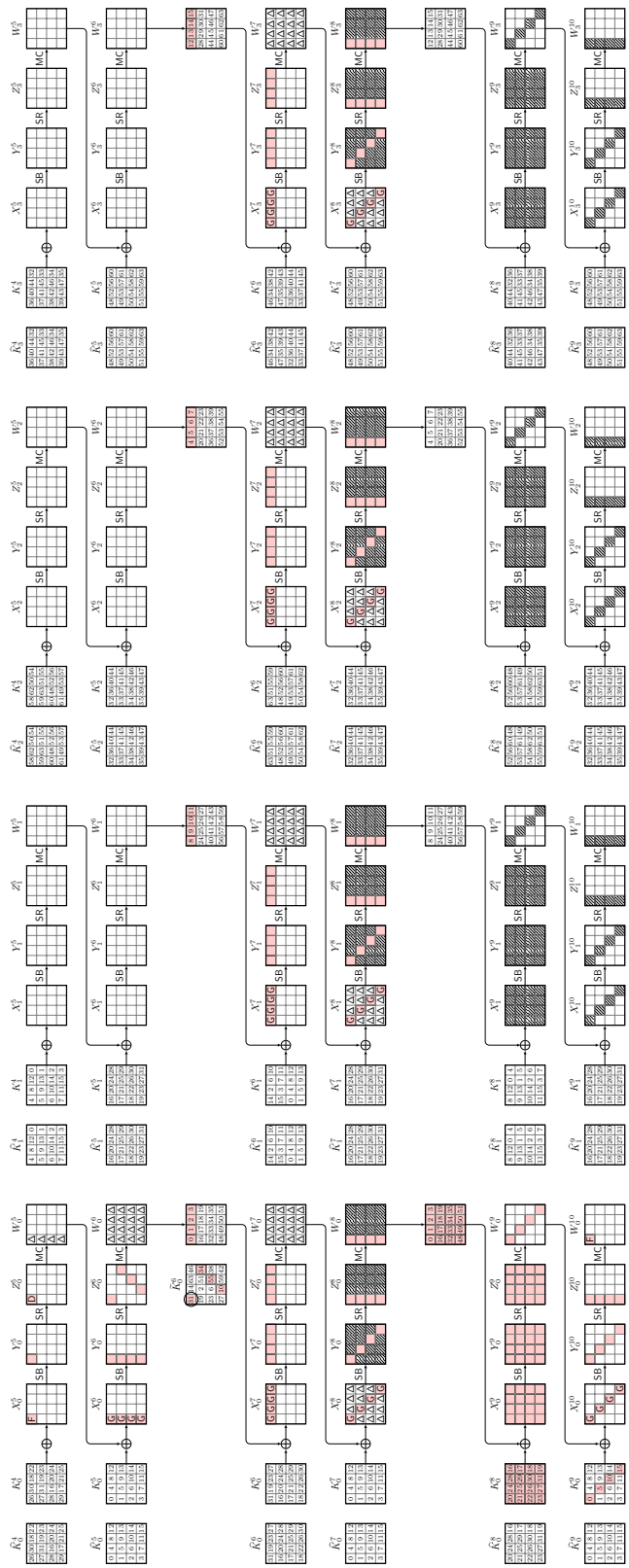


Figure 55: The offline distinguisher in the attack on 12-round Vistrutah-512.

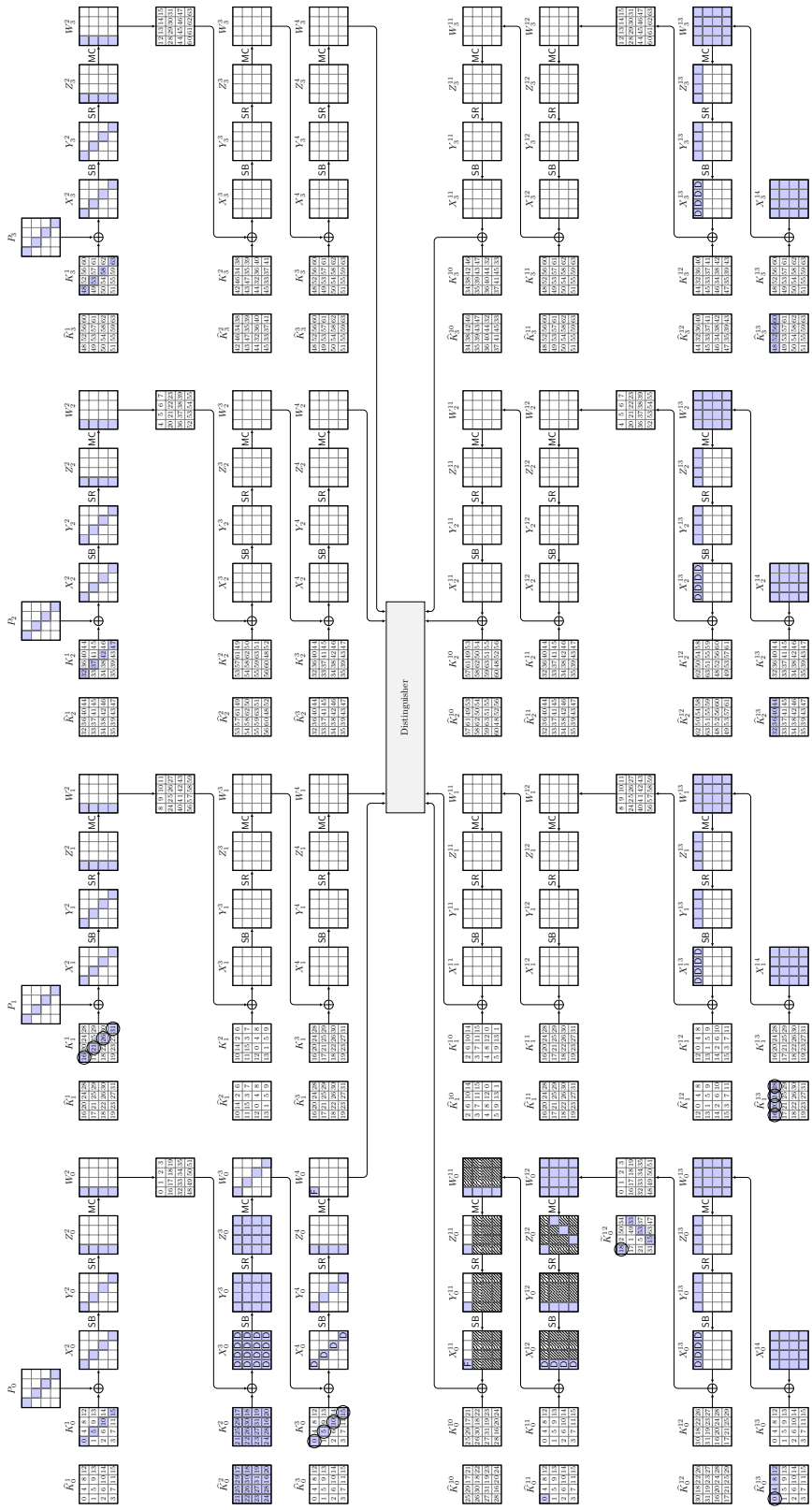


Figure 56: The outer rounds of the online filtering and key-recovery phase for the 12-round attack on `Vistrutah`-512.

- $\widetilde{K}_0^6[0,7,10,13]$, where $\widetilde{K}^6$ is the variant of the key bytes of X^7 to produce W_0^6 with the inverse `MixColumns` operation applied to it, i.e., $\mathsf{MC}^{-1}(SK[31], SK[19], \ldots, SK[42])$ as shown in Figure 55.
- K_0^8, which is identical to $SK[16..31]$, and
- K_0^9, which yields $SK[0,5,10,15]$.

Among those, one byte overlaps and yields a constraint in the key schedule (the circled cell), namely $\widetilde{K}_0^6[0]$ can be derived from K_0^8 and must produce the same value. Thus, we can expect approximately $2^{c+40\cdot 8}$ trails on average.

We note that also the key bytes obtained from the distinguisher are stored into the offline table. In the offline phase, we compute the lead pair and 62 related texts through the 6-round distinguisher, which has $1+4+16+16+16+4=57$ active S-boxes that have to be computed compared to 768 S-boxes in the 12-round cipher. We store 20 key bytes and 63 byte differences to the lead text, i.e., 83 bytes in total, as an entry into a table $\mathcal{L}$.

For $c=4$, the time complexity is approximately $2^4 \cdot 2^{40\cdot 8} \cdot (2+62) \cdot 57/768 \approx 2^{326.2}$ EEs and $2^{4+40\cdot 8} = 2^{324}$ MAs to $\mathcal{L}$. The memory complexity is $2^4 \cdot 2^{40\cdot 8} \cdot 83/64 \approx 2^{324.4}$ states.

E.7.4.2 Online Phase

The key-recovery phase is shown in Figure 56. In that phase, we construct 2^s structures of 2^{128} plaintexts each, iterating over all values of the first diagonals in P_0, P_1, P_2, and P_3. Thus, each structure can form 2^{255} pairs. Aiming at 2^{540} pairs, we require $s=285$, i.e., 2^{413} CPs in total.

We ask the oracle for all corresponding ciphertexts, inverts the final `MixColumns` operation, and stores all resulting texts of the current structure in a table $\mathcal{C}$. We identify pairs with only (at most) $\Delta Z_i^{13}[0,4,8,12]$ active for all $i \in \{0,1,2,3\}$. This has probability approximately $2^{-96\cdot 4} = 2^{-384}$. Thus, we expect $2^{540-384} = 2^{156}$ pairs to survive on average.

For each surviving pair, we would have to guess 40 bytes in addition to the 2^c allowed differences:

- ΔX_0^3,
- $\Delta X_0^4[0,5,10,15]$,
- $\Delta X_0^{12}[0,1,2,3]$, and
- $\Delta X_i^{13}[0,4,8,12]$ for $i \in \{0,1,2,3\}$.

For any given nonzero differential $(\Delta_{\text{in}}, \Delta_{\text{out}})$ there exists on average one pair of value that fulfills it for the `AES` S-box. From guessing the 40 bytes for a candidate plaintext (and ciphertext) pair, we can obtain the necessary state values from the pair of in- and output difference and hence derive the secret key bytes $SK[0, 5, 10, 15, 16..32, 37, 42, 47, 148, 53, 58, 63]$ and several key values with the inverse `MixColumns` operation applied to it. Among those, we can identify 14 key constraints, i.e., values that are defined by two equations:

- $K_1^1[0,5,10,15]$ must match $SK[16,21,26,31]$,
- $K_0^3[0,5,10,15]$ must match $SK[0,5,10,15]$,
- $\widetilde{K}_0^{12}[0]$, which maps the inputs from the state X^{13} to Z_0^{12}, must match four bytes that can be computed from $SK[18,17,21,31]$, which itself is known from K_0^2,
- $\widehat{K}_0^{11}[0]$ and $\widehat{K}_0^{13}[0]$ must match.
- $\widehat{K}_1^{13}[0,4,8,12]$ must match four bytes that can be computed from $SK[16,20,24,28]$, which is known from K_0^2.

The key-recovery part has 57 active S-boxes:

- 16, 16, and four active S-boxes in Rounds 2, 3, and 4, respectively, and
- one, four, and 16 active S-boxes in Rounds 10 through 12, respectively.

The attack proceeds as follows. For each surviving plaintext pair:

1. Guess ΔX_0^3 and $\Delta X_0^4[0,5,10,15]$, and compute both texts to $W_0^4[0]$.
2. There is an eight-byte filter from the four circled key constraints in K_1^1 and K_0^3 in Figure 56; filter all but the satisfying $2^{(20-8)\cdot 8} = 2^{96}$ candidates for each pair.
3. Form the related texts at $W_0^4[0]$, obtain their plaintexts, and look into $\mathcal{C}$ for their corresponding ciphertexts.

(Note that $16+16+4 = 36$ S-boxes are active in the first three partially computed rounds.) For each of the ciphertext pairs just obtained, the 2^{96} suggested key candidates, and the resulting related ciphertexts:

1. Guess 12 bytes of $\widehat{K}_i^{13}[0,4,8,12]$ for $i \in \{0,2,3\}$; note that it already knows $\widehat{K}_1^{13}[0,4,8,12]$.
2. Compute W_0^{12} and invert the `MixColumns` operation to obtain $\widehat{W}_0^{12}$.
3. Guess three more bytes $\widetilde{K}_0^{12}[7,10,13]$ ($\widetilde{K}_0^{12}[0]$ is known from K_0^2) and compute $X_0^{12}[0,1,2,3]$.
4. Using $\widetilde{K}_0^{11}[0] = \widehat{K}_0^{13}[0]$, compute $X_0^{11}[0]$, check if the pair is among the 2^c valid ones.
5. Derive the full states from the ciphertexts to $X_0^{11}[0]$ for all related texts and looks up the resulting set of differences in the offline table.
6. If any candidate exists, extract the key bytes from it, merge these with the key candidates of the entry (all 20 key bytes $SK[0,5,10,15,16,\ldots,31]$ in an offline entry can be compared with those of an online candidate) to obtain $20+23 = 43$ key bytes. Of these bytes, 20 are contained in both, and 23 more come from K_2^1, K_3^1, $\widehat{K}_2^{13}$, $\widehat{K}_3^{13}$, $\widehat{K}_0^{11}[0]$, $\widetilde{K}_0^{12}[7,10,13]$, and $\widehat{K}_0^{13}[4,8,12]$.
7. For each match, guess the 21 remaining key bytes with a full encryption each and output the key candidates that survive that test.

The data-collection costs 2^{413} full encryptions, and $2 \cdot 2^{413} = 2^{414}$ MAs to $\mathcal{C}$; The computational complexity of the online phase consists mainly of

- $2^{156} \cdot 2 \cdot 2^{c+20\cdot 8} \cdot 36/768 \approx 2^{316.6}$ EEs for the first pair through the first three rounds,
- $2^{156} \cdot 64 \cdot 2^{c+12\cdot 8} \cdot 36/768 \approx 2^{257.6}$ encryptions on average for the related texts, given that the first filter has eight byte constraints that serve as a filter,
- $2^{156} \cdot 64 \cdot 2^{c+12\cdot 8} = 2^{262}$ look ups into $\mathcal{C}$ to find the corresponding ciphertexts,
- $2^{156} \cdot 64 \cdot 2^{c+12\cdot 8} \cdot 2^{15\cdot 8} \cdot 21/768 \approx 2^{376.8}$ encryptions for the $12+3$ unknown guessed key bytes, 12 in $\widehat{K}_i^{13}$ and three in $\widetilde{K}_0^{12}$, for computing all texts until X_0^{11},
- $2^{156} \cdot 2^{c+(12+15)\cdot 8} = 2^{376}$ lookups into the offline table,
- $2^{376} \cdot 2^{324} \cdot 2^{-83\cdot 8} \approx 2^{36}$ expected matches, and
- $2^{36} \cdot 2^{21\cdot 8} = 2^{204}$ EEs to find the full key.

Those sum up to approximately $2^{376.8}$ EEs and 2^{376} lookups into huge tables.

E.7.4.3 Total Complexity

In total, the attack needs approximately

- Data-collection phase: 2^{413} EEs, 2^{414} MAs, 2^{413} CPs, and 2^{128} states of memory.
- Offline phase: $2^{326.2}$ EEs, 2^{324} MAs, and $2^{324.4}$ states of memory.
- Online phase: $2^{376.8}$ EEs and 2^{376} MAs.

We approximate the time complexity by the sum of encryptions and lookups, using one full encryption each for the latter to $2^{382.6}$ EEs.

E.7.5 Data Reduction

Note that there is a variant of the attack that trades higher computational complexity for a reduction in the data. For that purpose, one can use an offline phase with a difference that has a single active byte only in ΔW_0^{10} and allows ΔW_i^{10} for $i \in \{1, 2, 3\}$ to be fully active. The offline computational complexity is not affected. The attack needs a trail that occurs with probability 2^{-128} for having only a few allowed differences (a subset of the 2^c ones) occuring at ΔX_0^5 and 2^{-128} to have a certain difference in ΔW_0^{10}, i.e., approximately $2^{c-128-128} = 2^{c-256} = 2^{-252}$ for $c = 2^4$. Then, a single structure of 2^{128} CPs that can form 2^{255} pairs will suffice and will contain also the related texts of each pair.

The price is that the 2^{255} pairs cannot be filtered further, but will undergo efforts of 2^{160} partial computations at the plaintext side, which will produce 2^{96} key candidates on average, i.e., cost $2^{255} \cdot 64 \cdot 2^{160} \cdot 36/768 \approx 2^{416.6}$ EEs. At the ciphertext side, we have to guess at least 15 key bytes, which will add to $2^{255} \cdot 2^{96} \cdot 2^{120} \cdot 64 \cdot 21/768 \approx 2^{471.8}$ EEs and 2^{471} MAs. Therefore, this attack requires about:

- Data-collection phase: 2^{128} EEs, 2^{129} MAs, 2^{128} CPs, and 2^{128} states of memory.
- Offline phase: $2^{326.2}$ EEs, 2^{324} MAs, and $2^{324.4}$ states of memory.
- Online phase: $2^{471.8}$ EEs and 2^{471} MAs.