Partial Sums Meet FFT: Improved Attack on 6-Round AES

Orr Dunkelman[1], Shibam Ghosh[1(✉)], Nathan Keller[2], Gaëtan Leurent[3(✉)], Avichai Marmor[2], and Victor Mollimard[1]

[1] Computer Science Department, University of Haifa, Haifa, Israel
orrd@cs.haifa.ac.il, sghosh03@campus.haifa.ac.il
[2] Department of Mathematics, Bar Ilan University, Ramat Gan, Israel
Nathan.Keller@biu.ac.il, avichai@elmar.co.il
[3] Inria, Paris, France
gaetan.leurent@inria.fr

Abstract. The *partial sums* cryptanalytic technique was introduced in 2000 by Ferguson et al., who used it to break 6-round AES with time complexity of 2^{52} S-box computations – a record that has not been beaten ever since. In 2014, Todo and Aoki showed that for 6-round AES, partial sums can be replaced by a technique based on the Fast Fourier Transform (FFT), leading to an attack with a comparable complexity.

In this paper we show that the partial sums technique can be combined with an FFT-based technique, to get the best of the two worlds. Using our combined technique, we obtain an attack on 6-round AES with complexity of about $2^{46.4}$ additions. We fully implemented the attack experimentally, along with the partial sums attack and the Todo-Aoki attack, and confirmed that our attack improves the best known attack on 6-round AES by a factor of more than 32.

We expect that our technique can be used to significantly enhance numerous attacks that exploit the partial sums technique. To demonstrate this, we use our technique to improve the best known attack on 7-round Kuznyechik by a factor of more than 80.

1 Introduction

The *partial sums* cryptanalytic technique was introduced by Ferguson et al. [21] as a tool for enhancing the Square attack [16] on AES [1]. Informally, the Square attack requires computing the XOR of 2^{32} 8-bit values extracted from partially decrypted ciphertexts under each of 2^{40} candidate subkeys, which amounts to 2^{72} operations. The partial sums technique divides the attack into several steps where at each step, the adversary guesses several key bits and computes a 'partial sum', which allows reducing the number of partially decrypted values whose

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/978-3-031-58716-0_5.

M. Joye and G. Leander (Eds.): EUROCRYPT 2024, LNCS 14651, pp. 128–157, 2024.
https://doi.org/10.1007/978-3-031-58716-0_5

XOR should be computed. As a result, the overall complexity of the attack is significantly reduced to 2^{52} operations.

In the 23 years since the introduction of the partial sums technique, it was shown to enhance not only the Square attack but also several other attacks (e.g., integral, linear, zero-correlation linear, and multi-set algebraic attacks, see [4,6,8,12,17]) in various scenarios, and was applied to attack numerous ciphers (AES, Kuznyechik, MISTY1, CLEFIA, Skinny, Zorro, Midori, and LBlock, to mention a few). Yet, its best known application remained the original one – the attack on 6-round AES which remained the best attack on 6-round AES, despite many attempts to supersede it (see Table 2).

In 2014, Todo and Aoki [30] showed that an FFT-based technique can replace partial sums in enhancing the Square attack. The idea is to represent the XOR of the 2^{32} partially decrypted ciphertexts which the adversary has to compute as a *convolution* of two tailor-made functions and then to use the Fast Fourier Transform (FFT) in order to compute this value for all guessed subkeys at once, at the cost of about $4 \cdot 2^{32} \cdot \log(2^{32})$ addition operations. While at a first glance, this technique seems clearly advantageous over partial sums, subtle practical difficulties counter its advantages, making the two techniques comparable. First, the technique can be applied only after guessing 8 bits of the key. Secondly, as the output of the FFT is an element in $\mathbb{Z}$ and not an element in the finite field $GF(2^8)$, one has to repeat the procedure for each of the 8 bits in which the XOR should be computed. Thirdly, while partial sums can exploit partial knowledge of the subkeys the adversary needs to guess, it seems that the FFT-based technique does not gain anything from partial knowledge. According to the authors of [30], the complexity of their attack on 6-round AES is $6 \cdot 2^{50}$ addition operations, which is roughly equal to the complexity of the partial sums attack.

In the last decade, the Todo-Aoki technique was used as a comparable alternative of partial sums, with several authors mentioning advantages of each attack technique in different scenarios (see [4,6,15,32]). Yet, it seemed that one has to choose between the benefits of the two techniques in each application.

In this paper we show that one can combine partial sums with an FFT-based technique, getting the best of the two worlds in many cases. The basic idea behind our technique is to use the general structure of partial sums, but to replace particular key-guessing steps used in partial sums (or combinations of several such steps) by FFT-based steps, which include embedding finite field elements into $\mathbb{Z}$. We show that this allows computing the XOR in all 8 output bits at once, exploiting partial key knowledge, and even *packing* several computations together in the same 64-bit word addition and multiplication operations. As a result, we obtain the speedup of FFT over key guessing, without the disadvantages it carries in the Todo-Aoki technique. In addition, the new technique allows for much more flexibility, as we may choose which steps we group together and in which steps we use FFT instead of key guessing. The choice depends on multiple step-dependent parameters, such as the number of subkey bits guessed in the step, the ability to pre-compute some of the operations required for the FFT, and partial knowledge of subkey bits. Thus, the flexibility may be very helpful.

Table 1. Cost comparison of three best attacks on 6-Round AES in Amazon's AWS

Attack (Source)	AWS Instance	Running Time (in minutes)	Total Cost (in US$)
Square & Partial sums [21]	m6i.32xlarge	4859	497
Square & FFT [30]	r6i.32xlarge	3120	418
Square & Partial sums & FFT (Sect. 3.5)	m6i.32xlarge	48	5

We use our technique to mount an improved attack on 6-round AES. We obtain an attack which requires 2^{33} chosen plaintexts (compared to $2^{34.5}$ in the partial sums attack of [21]), time complexity of about $2^{46.4}$ additions (compared to 2^{52} S-box computations in partial sums), and memory complexity of 2^{27} 128-bit blocks (roughly the same as in partial sums). As it is hard to compare additions with S-box applications, we experimentally compared the attacks by fully implementing our attack, the partial sums attack, and the Todo-Aoki attack, using Amazon AWS servers. We optimized the instance which best fits the attacks (optimizing for performance/cost tradeoff). Our experiments show that our attack takes 48 minutes (and costs 5 US$), the partial sums attack takes 4859 minutes (and costs 497 US$), and the Todo-Aoki attack takes 3120 minutes (and costs 418 US$). Thus, our attack provides a speedup by a factor of more than 65 over both the partial sums attack and Todo-Aoki's attack, and allows breaking 6-round AES in about 48 minutes at the cost of only 5 US$. This breaks a 23-year old record in practical attacks on 6-round AES. Table 1 summarizes the costs of running the attacks. The source code is publicly available at the following link

https://github.com/ShibamCrS/Partial_Sums_Meet_FFT.

Our attack improves the partial sums attack of [21] on 7-round AES by the same factor. In addition, it might be applicable to other primitives that use 6-round AES as a component like the tweakable block cipher TNT-AES [5].

Due to the flexibility of our technique, it can be used to improve various attacks that use the partial sums technique. We demonstrate this applicability by presenting improved attack on Kuznyechik [18] the Russian Federation encryption standard. The best-known attack on Kuznyechik is a multiset-algebraic attack on 7 rounds (out of 9) with the complexity of $2^{154.5}$ encryptions, presented by Biryukov et al. [12]. We show that this attack can be improved by a factor of more than 80 to about 2^{148} encryptions, thus providing the best-known attack on Kuznyechik. A comparison of our results on 6-round AES and reduced Kuznyechik with previously known results is presented in Table 2.

The full version of this paper [19] presents our techniques with two other targets MISTY1 and CLEFIA. We improve the Bar-On and Keller [8] attack by a factor 6 (to 2^{67}) and obtain the best known attacks against full MISTY1. We also improved multiple attacks against CLEFIA [13,23,28] for 11, 12, and 14 rounds. Most strikingly, we improve the 12-round attack of Sasaki and Wang [28] by a factor of about 2^{30}.

Table 2. Comparison of our results with previous key recovery attacks on 6-Round AES and reduced Kuznyechik. The results are listed in chronological order.

Cipher	Rounds	Data	Time	Technique and Source
AES	6	2^{32} CP	2^{71} Enc.	Square [16]
		$6 \cdot 2^{32}$ CP	2^{52} S-box Eval.	Square & Partial sums [21]
		2^{71} ACPC	2^{71} Enc.	Boomerang [11]
		2^{33} CP	2^{52} S-box Eval.	Square & Partial sums [31]
		$6 \cdot 2^{32}$ CP	2^{52} Add.	Square & FFT [30]
		2^{26} CP	2^{80} Enc.	Mixture Differential [7]
		2^{55} ACPC	2^{80} Enc.	Retracing Boomerang [20]
		$2^{79.7}$ ACPC	2^{78} Enc.	Boomeyong [27]
		2^{59} ACPC	2^{61} Enc.	Truncated Boomerang [9]
		2^{33} CP	$2^{46.4}$ Add.	Square & Partial sums & FFT (Sect. 3)
Kuznyechik	7	2^{128} KP	$2^{154.5}$ Enc.	Integral & Partial sums [12]
		2^{128} KP	2^{148} Enc.	Integral & Partial sums & FFT (Sect. 4)
	6	2^{120} CP	$2^{146.5}$ Enc.	Integral & Partial sums [12]
		2^{120} CP	$2^{140.9}$ Enc.	Integral & Partial sums & FFT (Sect. 4)

The paper is organized as follows. In Sect. 2, we describe the structure of the AES, the Square attack, and the two previously known methods for enhancing it – partial sums and the Todo-Aoki FFT-based method. Section 3 presents our new technique, along with its application to 6-round AES. Section 4 presents application of the new technique to the cipher Kuznyechik.

2 Background

2.1 Description of AES

AES [1] is a 128-bit block cipher, designed by Rijmen and Daemen in 1997 (originally, under the name Rijndael). In 2001, it was selected by the US National Institute of Standards (NIST) as the Advanced Encryption Standard, and since then, it has gradually become the most widely used block cipher worldwide.

AES is a Substitution-Permutation Network operating on a 128-bit state organized as a 4×4 array of 8-bit words. The encryption process is composed of 10, 12, or 14 rounds (depending on the key length: 10 rounds for 128-bit keys, 12 rounds for 192-bit keys, and 14 rounds for 256-bit keys). Each round of AES is composed of four operations, presented in Fig. 1.

SubBytes. Apply a known 8-bit S-box independently to the bytes of the state;
ShiftRows. Shift each row of the state to the left by the position of the row;
MixColumns. Multiply each column by the same known invertible 4-by-4 matrix over the finite field $GF(2^8)$;
AddRoundKey. Add a 128-bit round key computed from the secret key to the state, using a bitwise XOR operation.

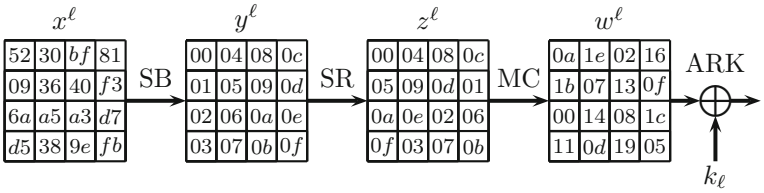

Fig. 1. An AES Round

An additional AddRoundKey operation is applied before the first round, and the last MixColumns operation is omitted. As properties of the key schedule of AES are not used in this paper, we refer the reader to [1] for its description.

The rounds are numbered from 0 to $Nr - 1$, where Nr is the number of rounds. The subkey used in the AddRoundKey operation of round ℓ is denoted by k^ℓ, and the j'th byte in its i'th row is denoted by k^ℓ_{4j+i}. The whitening key added before the initial round is denoted by k^{-1}. The j'th byte in the i'th row of the state before the SubBytes, ShiftRows, MixColumns, AddRoundKey operations of round ℓ is denoted by $x^\ell_{4j+i}, y^\ell_{4j+i}, z^\ell_{4j+i}$, and w^ℓ_{4j+i}, respectively. A set of bytes $\{v_i, v_j, v_k\}$ is denoted by $v_{i,j,k}$.

2.2 The Square Attack on AES

AES was designed as a modification of the block cipher Square [16], which came together with a dedicated attack, called 'the Square attack'. This attack, in its basic application to AES, uses the following observation.

Lemma 1. *Consider the encryption by 3-round AES of a set of 256 plaintexts, $P_0, P_1, \ldots, P_{255}$, which are equal in all bytes except for a single byte, such that the single byte assumes each possible value exactly once. Then the corresponding ciphertexts $C_0, C_1, \ldots, C_{255}$ satisfy $\bigoplus_{i=0}^{255} C_i = 0$.*

As was shown in [16], this property can be used to attack 6-round Square, and also 6-round AES, with a complexity of about 2^{80} S-box computations. The adversary asks for the encryption of 2^{32} plaintexts which are equal in all bytes except for the main diagonal (i.e., bytes 0,5,10,15) and assume all 2^{32} possible values in the main diagonal. Then, he guesses bytes $0, 5, 10, 15$ of k^{-1}, and for each guess, he partially encrypts the plaintexts through round 0 and finds a set of 2^8 inputs to round 1 which satisfy the assumption of Lemma 1. Then, he partially guesses the subkeys k^4, k^5, partially decrypts the 2^8 corresponding ciphertexts through rounds 4,5 and checks whether the XOR of the 2^8 corresponding values at the state x^4_0 (i.e., at byte 0 before the SubBytes operation of round 4) is zero, as is stated by Lemma 1. If not, the subkey guess is discarded.

While it seems that in order to compute byte x^4_0 from the ciphertext, the adversary must know 64 subkey bits (specifically, key bytes $k^5_{0,7,10,13}$ and $k^4_{0,1,2,3}$), in fact knowing 40 subkey bits is sufficient. Indeed, since MixColumns is a linear operation, it can be interchanged with the AddRoundKey operation after it, at the cost of replacing k^4 with the equivalent subkey $\bar{k}^4 = \text{MixColumns}^{-1}(k^4)$.

The knowledge of the key bytes $k^5_{0,7,10,13}$ and $\bar{k}^4_0$ is sufficient for computing the state byte x^4_0 from the ciphertext of 6-round AES.[1] Each check whether 2^8 values XOR to zero provides an 8-bit filtering, and hence, checking several sets is sufficient for discarding all wrong subkey guesses. The attack recovers 9 subkey bytes $(k^{-1}_{0,5,10,15}, \bar{k}^4_0, k^5_{0,7,10,13})$ with complexity of about $2^{32} \cdot 2^{40} \cdot 2^8 = 2^{80}$ S-box computations.

In [21], Ferguson et al. observed that the Square attack can be improved by replacing Lemma 1 with the following lemma on 4-round AES.

Lemma 2. *Consider the encryption by 4-round AES of a set of 2^{32} plaintexts, $P_0, P_1, \ldots, P_{2^{32}-1}$, which are equal in all bytes except for the main diagonal (i.e., bytes 0,5,10,15), such that the diagonal assumes each possible value exactly once. Then the corresponding ciphertexts $C_0, C_1, \ldots, C_{2^{32}-1}$ satisfy $\bigoplus_{i=0}^{2^{32}-1} C_i = 0$.*

Lemma 2 can be used to attack 6-round AES using the same strategy described above. The adversary asks for the encryption of a few sets of 2^{32} plaintexts which satisfy the assumption of Lemma 2. Then, for each set, he guesses subkey bytes $\bar{k}^4_0, k^5_{0,7,10,13}$ and checks whether the XOR of the 2^{32} intermediate values at the state byte x^4_0 is zero, as is stated by Lemma 2. The attack recovers 5 subkey bytes $(\bar{k}^4_0, k^5_{0,7,10,13})$ and its complexity is about $2^{32} \cdot 2^{40} = 2^{72}$ S-box computations.

2.3 The Partial Sums Attack

In the same paper [21], Ferguson et al. showed that the complexity of the Square attack described above can be significantly reduced, by dividing the key guessing and partial decryption into several steps and gradually reducing the number of values whose XOR should be computed. By the structure of AES, the state byte x^4_0 is computed from the ciphertext C using the following formula:

$$x^4_0 = S^{-1}\big(\bar{k}^4_0 \oplus \mathtt{0e_x} \cdot S^{-1}(C_0 \oplus k^5_0) \oplus \mathtt{09_x} \cdot S^{-1}(C_7 \oplus k^5_7) \oplus \\ \oplus \mathtt{0d_x} \cdot S^{-1}(C_{10} \oplus k^5_{10}) \oplus \mathtt{0b_x} \cdot S^{-1}(C_{13} \oplus k^5_{13})\big), \tag{1}$$

where the coefficients $\mathtt{0e_x}, \mathtt{09_x}, \mathtt{0d_x}, \mathtt{0b_x}$ come from the inverse MixColumns operation and the multiplication is performed in the finite field $GF(2^8)$.

Note that the right hand side of (1) depends only on bytes 0,7,10,13 of the ciphertext. This means that if two ciphertexts are equal in these four bytes, then their contributions to the XOR of x^4_0 values cancel each other. Thus, we may replace the list of ciphertexts with a list A of 2^{32} binary indices which indicates whether each of the 2^{32} possible values of bytes 0,7,10,13 of the ciphertext appears an even or an odd number of times in the list of ciphertexts. The goal of the subsequent steps is to reduce the number of needed binary indices, in parallel to guessing subkey bytes.

[1] Here and in the sequel, we assume that in 6-round AES, the MixColumns operation of round 5 is omitted. If this operation is not omitted, the attack works almost without change; we only have to replace the key k^5 with the equivalent key $\bar{k}^5 = \text{MixColumns}^{-1}(k^5)$.

At the first step, the adversary guesses bytes 0,7 of k^5, and reduces the size of the list to 2^{24}. Denote $a_1 = \mathtt{0e_x} \cdot S^{-1}(C_0 \oplus k_0^5) \oplus \mathtt{09_x} \cdot S^{-1}(C_7 \oplus k_7^5)$. Observe that if two ciphertexts are equal in the bytes a_1, C_{10}, C_{13}, then their contributions to the XOR of x_0^4 values cancel each other. As the guess of bytes $k_{0,7}^5$ allows computing a_1 for each ciphertext, the adversary can construct a list A_1 of 2^{24} binary values which indicates whether each possible value of (a_1, C_{10}, C_{13}) appears an even or an odd number of times in the list of intermediate values. The complexity of this step is about $2^{16} \cdot 2^{32} = 2^{48}$ S-box evaluations.

At the second step, the adversary guesses the byte k_{10}^5 and reduces the list to a list A_2 of size 2^{16} that corresponds to the possible values of (a_2, C_{13}), where $a_2 = a_1 \oplus \mathtt{0d_x} \cdot S^{-1}(C_{10} \oplus k_{10}^5)$. At the third step, the adversary guesses the byte k_{13}^5 and reduces the list to a list A_3 of size 2^8 that corresponds to the possible values of a_3, where $a_3 = a_2 \oplus \mathtt{0b_x} \cdot S^{-1}(C_{13} \oplus k_{13}^5)$. Finally, at the fourth step, the adversary guesses the byte $\bar{k}_0^4$, computes $\oplus_{\{x \in \{0,1\}^8 : A_3[x]=1\}} S^{-1}(\bar{k}_0^4 \oplus x)$, which is equal to the right hand side of (1), and checks whether it is equal to zero. The complexity of each step is about 2^{48} S-box computations, and thus, the overall complexity for a single set of 2^{32} plaintexts is 2^{50} S-box computations.

As the attack recovers 5 subkey bytes, six sets of 2^{32} plaintexts are required to recover their value uniquely with a high probability. Note that after the check of the first set, only about $2^{40} \cdot 2^{-8} = 2^{32}$ suggestions for the 40 subkey bits remain undiscarded. This means that for each possible value of $k_{0,7,10,13}^5$, at most a few values of $\bar{k}_0^4$ that correspond to them are expected to remain. Hence, when examining the second set of 2^{32} plaintexts, the complexity of the fourth step becomes negligible as it is performed only for a few values of $\bar{k}_0^4$. Similarly, when examining the third set, the two last steps become negligible, etc. In total, the complexity of checking all six plaintext sets of size 2^{32} is equivalent to the cost of $4 + 3 + 2 + 1 = 10$ steps, or $2^{51.3}$ S-box computations.[2]

The attack is given as Algorithm 1. To simplify the notation, we rewrite equation (1) in a more generic way, using S_0 for $\mathtt{0e_x} \cdot S^{-1}(\cdot)$, S_1 for $\mathtt{09_x} \cdot S^{-1}(\cdot)$, S_2 for $\mathtt{0d_x} \cdot S^{-1}(\cdot)$, S_3 for $\mathtt{0b_x} \cdot S^{-1}(\cdot)$, and renaming the keys and the ciphertext bytes to k_0, k_1, k_2, k_3, k_4 and c_0, c_1, c_2, c_3, respectively:

$$a_4 = S^{-1}\left(k_4 \oplus S_0(c_0 \oplus k_0) \oplus S_1(c_1 \oplus k_1) \oplus S_2(c_2 \oplus k_2) \oplus S_3(c_3 \oplus k_3)\right). \quad (2)$$

Reducing the Data Complexity. In [31], Tunstall observed that the data complexity of the attack can be reduced to 2^{33} chosen plaintexts by examining two sets of 2^{32} plaintexts instead of six sets. The idea is to check an analogue of Eq. (1) for three additional bytes – x_5^4, x_{10}^4, and x_{15}^4 – using the same set of 2^{32} plaintexts. Note that in order to compute each of these three bytes from the ciphertext, the adversary needs the subkey bytes $k_{0,7,10,13}^5$ (which are the same as in Eq. (1)), along with a different byte of $\bar{k}^4$. When two sets are checked at

[2] We note that in [21], the authors performed a similar analysis and concluded that the complexity is 2^{52} S-box computations. This value was used in all subsequent papers. For the sake of consistency, we use the same value in Table 2, but note that the actual complexity is lower, as is shown here, and use the lower estimate when comparing the partial sums attack with our new attack.

Algorithm 1. Partial-sum algorithm for key recovery [21].

Input: Array A of bits such that the j^{th} value of A denotes the parity of the number of occurrences of j in the list of ciphertexts
for all k_0, k_1 **do**
  Declare an empty bit-array A_1 of size 2^{24}
  for all c_0, c_1, c_2, c_3 **do**
    if $A[c_0, c_1, c_2, c_3] = 1$ **then**
      $a_1 \leftarrow S_0(c_0 \oplus k_0) \oplus S_1(c_1 \oplus k_1)$
      $A_1[a_1, c_2, c_3] \leftarrow A_1[a_1, c_2, c_3] \oplus 1$
  for all k_2 **do**
    Declare an empty bit-array A_2 of size 2^{16}
    for all a_1, c_2, c_3 **do**
      if $A_1[a_1, c_2, c_3] = 1$ **then**
        $a_2 \leftarrow a_1 \oplus S_2(c_2 \oplus k_2)$
        $A_2[a_2, c_3] \leftarrow A_2[a_2, c_3] \oplus 1$
    for all k_3 **do**
      Declare an empty bit-array A_3 of size 2^8
      for all a_2, c_3 **do**
        if $A_2[a_2, c_3] = 1$ **then**
          $a_3 \leftarrow a_2 \oplus S_3(c_3 \oplus k_3)$
          $A_3[a_3] \leftarrow A_3[a_3] \oplus 1$
      for all k_4 **do**
        $a_4 \leftarrow 0$
        for all a_3 **do**
          if $A_3[a_3] = 1$ **then**
            $a_4 \leftarrow a_4 \oplus S^{-1}(k_4 \oplus a_3)$
        if $a_4 \neq 0$ **then**
          k_0, k_1, k_2, k_3, k_4 is not a valid key candidate

the same byte, they provide a 16-bit filtering, which in particular yields an 8-bit filtering on the value $k^5_{0,7,10,13}$ which is common to all examined bytes. Hence, information from different bytes can be combined to recover $k^5_{0,7,10,13}$ with a high probability.

The data complexity can be further reduced to 2^{32} by examining a single set and checking the XOR in all 16 bytes of x^4. The algorithm is more complex and uses a meet-in-the-middle procedure based on the properties of the AES key schedule. We omit the description here, as it will not be needed in the sequel.

In [31], it is claimed that when the same set of plaintexts is used to check the parity in several bytes, the complexity of checking the first byte is dominant, as some of the computations performed for computing the XOR in different bytes are identical. However, this claim seems incorrect, as in the variant of Eq. (1) for other bytes, the order of the coefficients $\mathtt{0e}_{\mathtt{x}}, \mathtt{09}_{\mathtt{x}}, \mathtt{0d}_{\mathtt{x}}, \mathtt{0b}_{\mathtt{x}}$ which stems from the inverse MixColumns operation is changed, and hence, the operations performed for different bytes are not identical and only knowledge of subkeys can be 'reused'. Therefore, the complexity of the attack that uses two sets is about $(4 + 3 + 2 + 2 + 1 + 1) \cdot 2^{48} = 2^{51.7}$ S-box computations, and the attack that uses one set takes about $16 \cdot 2^{50} = 2^{54}$ S-box computations.

The idea of using two sets of size 2^{32} instead of six was independently suggested in [2] by Alda et al., who also experimentally verified it.

2.4 The FFT-Based Attack of Todo and Aoki

The general idea of using the Fast Fourier Transform (FFT) for speeding up cryptanalytic attacks on block ciphers goes back to Collard et al. [14] who used the FFT to speed up linear cryptanalysis. This idea was extended to several other techniques, including multi-dimensional linear attacks [25,26], zero-correlation attacks [13], differential-linear attacks [10], etc.

In [30], Todo and Aoki proposed to replace the partial sums technique by an FFT-based technique. The basic idea behind the Todo-Aoki technique is that the sum of the values in the right hand side of Eq. (1) which we want to compute can be written in the form of a *convolution* of tailor-made functions, as seen in Algorithm 2.

Consider a set S of 2^{32} ciphertexts for which we want to compute the XOR of the intermediate values at the state byte x_0^4. Like in the partial sums attack, denote by A a bit array of size 2^{32}, such that $A(c_0, c_1, c_2, c_3) = 1$ if and only if $C_{0,7,10,13} = (c_0, c_1, c_2, c_3)$ holds for an odd number of ciphertexts in S. Let $f : \{0,1\}^{32} \to \{0,1\}$ be the indicator function of the array, that is, $f(c_0, c_1, c_2, c_3) = \mathbb{1}(A(c_0, c_1, c_2, c_3) = 1)$. Assume that the subkey k_4 was guessed, and let $g_i : \{0,1\}^{32} \to \{0,1\}$, for $0 \le i \le 7$, be defined by

$$g_i(t_0, t_1, t_2, t_3) = \left[S^{-1}\left(k_4 \oplus S_0(t_0) \oplus S_1(t_1) \oplus S_2(t_2) \oplus S_3(t_3)\right)\right]_i, \tag{3}$$

where $[S^{-1}(t)]_i$ denotes the i'th bit of $S^{-1}(t)$. Then, denoting by $[x(C, k_0, k_1, k_2, k_3)]_i$ the i'th bit of the value x_0^4 corresponding to the ciphertext C for a given guess of k_0, k_1, k_2, k_3 (see Eq. (2)), we have

$$\begin{aligned}
\bigoplus_{C \in S} [x(C, k_0, k_1, k_2, k_3)]_i &= \bigoplus_{\{(c_0,c_1,c_2,c_3) : A[c_0,c_1,c_2,c_3]=1\}} g_i(c_0 \oplus k_0, c_1 \oplus k_1, c_2 \oplus k_2, c_3 \oplus k_3) \\
&= \bigoplus_{c_0,c_1,c_2,c_3} f(c_0, c_1, c_2, c_3) \cdot g_i(c_0 \oplus k_0, c_1 \oplus k_1, c_2 \oplus k_2, c_3 \oplus k_3) \\
&= (f * g_i)(k_0, k_1, k_2, k_3).
\end{aligned}$$

Therefore, we can compute the sum for all 2^{32} possible guesses of (k_0, k_1, k_2, k_3) at once by guessing the byte k_4 and computing the convolution of two functions on 32 bits, that takes time of about $4 \cdot 2^{32} \log_2(2^{32})$ additions, as was shown by Collard et al. [14]. As the summation is performed for each bit separately, the complexity of examining a single set S of 2^{32} ciphertexts is $8 \cdot 2^8 \cdot 4 \cdot 2^{32} \log_2(2^{32}) = 2^{50}$ additions, which is roughly equal to the number of operations required for examining a single set of ciphertexts in the partial sums attack.

A disadvantage of the Todo-Aoki technique, compared to the partial sums attack, is that it cannot use partial knowledge of the subkey to obtain a speedup. Indeed, as the computation is performed for all values of (k_0, k_1, k_2, k_3) at the same time, partial knowledge (e.g., knowledge of k_3) cannot be exploited. As

Algorithm 2. FFT-based algorithm for key recovery [30]. The blue colored step has naive complexity $2^{32} \times 2^{32}$, but can be replaced by several Hadamard transformations of size 2^{32} with complexity 2^{37} each.

Input: Array A of bits such that the j^{th} value of A denotes the parity of the number of occurrences of j in the list of ciphertexts
for all k_4 **do**
  for all k_0, k_1, k_2, k_3 **do**
    $A_1[k_0, k_1, k_2, k_3] \leftarrow \bigoplus_{c_0,c_1,c_2,c_3} A[c_0, c_1, c_2, c_3] \cdot S^{-1}\begin{pmatrix} k_4 \oplus S_0(c_0 \oplus k_0) \oplus S_1(c_1 \oplus k_1) \\ \oplus S_2(c_2 \oplus k_2) \oplus S_3(c_3 \oplus k_3) \end{pmatrix}$
  for all k_0, k_1, k_2, k_3 **do**
    if $A_1[k_0, k_1, k_2, k_3] \neq 0$ **then**
      k_0, k_1, k_2, k_3, k_4 is not a valid key candidate

a result, when six sets of 2^{32} ciphertexts are examined, the complexity of the Todo-Aoki attack becomes $6 \cdot 2^{50} = 2^{52.6}$ additions, while the overall complexity of partial sums is only $2^{51.3}$ S-box computations, as was shown above.

The question, whether there is a way to use partial knowledge of the key in an FFT-based attack, was explicitly mentioned as an open question in [30].

Using Precomputation of the FFT to Speed Up the Attack. In the eprint version of the same paper [29], Todo showed that the complexity of the attack can be reduced by precomputing some of the Fast Fourier Transforms that should be computed in the course of the attack.

Recall that the computation of the convolution of $f, g : \{0,1\}^n \to \{0,1\}$ using the FFT consists of three stages:

1. Computing the Fourier transforms $\hat{f}, \hat{g} : \{0,1\}^n \to \mathbb{Z}$.
2. Computing the pointwise product $h : \{0,1\}^n \to \mathbb{Z}$ defined by $h(x) = \hat{f}(x) \cdot \hat{g}(x)$.
3. Computing the inverse Fourier transform (which is the same as computing the Fourier transform and dividing by 2^n) to obtain $f * g = \hat{h} \cdot 2^{-n}$.

Here, we use the convention that the Fourier transform $\hat{f}$ is obtained from f by writing f as a 2^n-dimensional vector and multiplying it by the Hadamard matrix H_n, defined recursively as $H_n = \begin{pmatrix} H_{n-1} & H_{n-1} \\ H_{n-1} & -H_{n-1} \end{pmatrix}$, where $H_1 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$.

The cost of each computation of the FFT is $n2^n$ addition operations. In order to avoid overflow the additions should have at least $2n$ bits of precision, but since we only want one bit of the result the computation can be done with $n+1$ bits of precision. For the 6-round AES attack we have $n = 32$ and the FFT will typically be implemented with 64-bit additions. The cost of the pointwise product is about 2^n multiplication operations, which is not much more than the cost of 2^n addition operations for small n (in particular for a software implementation with $n \leq 32$, as in the attack on 6-round AES).[3] Hence, the overall cost of the convolution computation in our case is about $3 \cdot 32 \cdot 2^{32}$ additions.

[3] We note that in [30], the authors conservatively estimate that pointwise multiplication of two vectors of size 2^n whose entries are n-bit integers takes $n2^n$ addition

Todo observed that the Fourier transforms $\hat{f}$ and $\hat{g}$ can be precomputed. As the function f does not depend on the guess of k_4, one can compute it once, store the result (which requires at most 2^{32} 64-bit words), and re-use it for each value of k_4. As the cost of this FFT computation is $32 \cdot 2^{32}$ additions, the amortization over guesses of k_4 makes it negligible. The function g cannot be precomputed since it depends on k_4. On the other hand, as it does not depend on the ciphertexts, it can be reused for other sets of ciphertexts. Therefore, the complexity of computing the XOR for a single set of 2^{32} ciphertexts is reduced to about $8 \cdot 2^8 \cdot 2 \cdot 32 \cdot 2^{32} = 2^{49}$ addition operations, and the complexity of computing the XOR for six sets is reduced to about $2^{49} + 5 \cdot 2^{48} = 2^{50.8}$ addition operations. If only two sets are examined and the XOR is computed in four bytes (as was described above), then the complexity becomes $2^{49} + 7 \cdot 2^{48} = 2^{51.2}$ addition operations. This complexity seems a bit lower than the complexity of partial sums, but it is still quite close and the different types of operations make comparison between the techniques tricky.

3 The New Technique: Partial Sums Meet FFT

In this section, we describe our new technique which allows combining the advantages of the partial sums technique with those of the Todo-Aoki FFT-based technique. We begin with a basic variant of the technique in Sect. 3.1, then we show how the complexity can be reduced significantly by packing several FFT computations together in Sect. 3.2, afterward, we present several additional enhancements and other variants of the basic technique in Sect. 3.3, and we conclude this section with a comparison of our technique with partial sums and the Todo-Aoki technique in Sect. 3.4. For the sake of concreteness, we present the attack in the case of 6-round AES and reuse the notations of Sect. 2. It will be apparent from the description how our technique can be applied in general.

3.1 The Basic Technique

Our basic observation is that we can follow the general structure of the partial sums attack, and replace each step by computing a convolution of properly chosen functions. This is shown in Algorithm 3 which is a rearrangement of the operations of Algorithm 1, making convolution appear. As we use somewhat different convolutions for different steps of the attack, we present them separately.

First Step. As described in Sect. 2.3, before the first step of the partial sums attack, the list of ciphertexts is replaced with a list A of 2^{32} binary indices which indicate whether each of the 2^{32} possible values of the bytes c_0, c_1, c_2, c_3 appears an even or an odd number of times in the list of ciphertexts. At the first step, the adversary guesses the bytes k_0, k_1, and replaces the list by a list A_1 of size 2^{24} which corresponds to the bytes a_1, c_2, c_3, where $a_1 = S_0(c_0 \oplus k_0) \oplus S_1(c_1 \oplus k_1)$.

operations. For the sake of consistency with [30] and fairness, we use the conservative estimate in the table of results and the less conservative estimate when we compare the Todo-Aoki technique to our technique.

We observe that the list A_1 can be computed for all values k_0, k_1 simultaneously by computing a convolution. Let $\chi : \{0,1\}^{32} \to \{0,1\}$ be the indicator function of the list A. That is, $\chi(c_0, c_1, c_2, c_3) = 1$ if and only if the value $(C_0, C_7, C_{10}, C_{13}) = (c_0, c_1, c_2, c_3)$ appears an odd number of times in the list of ciphertexts. For any $c_2, c_3 \in \{0,1\}^8$, define $\chi^1_{c_2,c_3}(c_0, c_1) = \chi(c_0, c_1, c_2, c_3)$.

For any $a_1 \in \{0,1\}^8$, let $I^1_{a_1}(x, y) = \mathbb{1}(S_0(x) \oplus S_1(y) = a_1)$. Both $\chi^1_{c_2,c_3}$ and $I^1_{a_1}$ are indicator functions on $\{0,1\}^{16}$. For any $a_1, c_2, c_3 \in \{0,1\}^8$, we have

$$\begin{aligned}(\chi^1_{c_2,c_3} * I^1_{a_1})(k_0, k_1) &= \sum_{c_0,c_1 \in \{0,1\}^8} \chi^1_{c_2,c_3}(c_0, c_1) \cdot I^1_{a_1}(c_0 \oplus k_0, c_1 \oplus k_1) \\ &= \sum_{c_0,c_1 \in \{0,1\}^8} \chi(c_0, c_1, c_2, c_3) \cdot \mathbb{1}(S_0(c_0 \oplus k_0) \oplus S_1(c_1 \oplus k_1) = a_1).\end{aligned}$$

Therefore, the entry which corresponds to (a_1, c_2, c_3) in the list $A_1[k_0, k_1]$ created for the subkey guess (k_0, k_1) is

$$A_1[k_0, k_1][a_1, c_2, c_3] = \left((\chi^1_{c_2,c_3} * I^1_{a_1})(k_0, k_1)\right) \bmod 2. \tag{4}$$

(Formally, we define A_1, which is a list of size 2^{24} that depends on two key bytes, as an array of size $2^{16} \times 2^{24}$ which includes the guessed bytes.) As was shown in Sect. 2.4, the computation of this convolution requires $3 \cdot 16 \cdot 2^{16}$ addition operations for each value of a_1, c_2, c_3, or a total of $48 \cdot 2^{40}$ additions. This compares favorably with the first step of the partial sums attack which requires 2^{48} S-box computations. As we shall see below, the actual advantage of our technique is significantly larger. However, this requires to store the full A_1 for all values of (k_0, k_1), of size 2^{40} bits.

Second Step. At the second step of the partial sums attack, the adversary guesses the byte k_2 and reduces the list A_1 to a list A_2 of size 2^{16} that corresponds to the possible values of (a_2, c_3), where $a_2 = a_1 \oplus S_2(c_2 \oplus k_2)$.

We compute the entries of the list A_2 using a convolution, as follows. For any $k_0, k_1, c_3 \in \{0,1\}^8$, define

$$\chi^2_{k_0,k_1,c_3}(a_1, c_2) = \mathbb{1}(A_1[k_0, k_1][a_1, c_2, c_3]) \qquad I^2(x, y) = \mathbb{1}(x = S_2(y)).$$

Both $\chi^2_{k_0,k_1,c_3}$ and I^2 are indicator functions on $\{0,1\}^{16}$. For any $k_0, k_1, c_3 \in \{0,1\}^8$, we have

$$\begin{aligned}(\chi^2_{k_0,k_1,c_3} * I^2)(a_2, k_2) &= \sum_{a_1,c_2 \in \{0,1\}^8} \chi^2_{k_0,k_1,c_3}(a_1, c_2) \cdot I^2(a_1 \oplus a_2, c_2 \oplus k_2) \\ &= \sum_{a_1,c_2 \in \{0,1\}^8} \mathbb{1}(A_1[k_0, k_1][a_1, c_2, c_3]) \cdot \mathbb{1}(a_1 \oplus a_2 = S_2(c_2 \oplus k_2)) \\ &= \sum_{a_1,c_2 \in \{0,1\}^8} \mathbb{1}(A_1[k_0, k_1][a_1, c_2, c_3]) \cdot \mathbb{1}(a_2 = a_1 \oplus S_2(c_2 \oplus k_2)).\end{aligned}$$

Therefore, the entry which corresponds to (a_2, c_3) in the list A_2 created for the subkey guess (k_0, k_1, k_2) is

$$A_2[k_2][a_2, c_3] = \left((\chi^2_{k_0,k_1,c_3} * I^2)(a_2, k_2)\right) \bmod 2. \tag{5}$$

(Note that formally, we define A_2, which is a list of size 2^{16} that depends on three key bytes, as an array of size $2^8 \times 2^{16}$, which depends on k_0, k_1). As above, the complexity of this step is $48 \cdot 2^{40}$ additions.

Third Step. This step is similar to the second step. Thus, we present it briefly. At the third step of the partial sums attack, the adversary guesses the byte k_3 and reduces the list A_2 to a list A_3 of size 2^8 that corresponds to the possible values of a_3, where $a_3 = a_2 \oplus S_3(c_3 \oplus k_3)$. We obtain the list A_3 by defining

$$\chi^3_{k_0,k_1,k_2}(a_2, c_3) = \mathbb{1}(A_2[k_2][a_2, c_3]) \qquad \text{and} \qquad I^3(x, y) = \mathbb{1}(x = S_3(y)),$$

and setting

$$A_3[k_3][a_3] = \left((\chi^3_{k_0,k_1,k_2} * I^3)(a_3, k_3)\right) \bmod 2. \tag{6}$$

(Note that formally, we define A_3 as an array of size $2^8 \times 2^8$, which depends on k_0, k_1, k_2). As above, the complexity of this step is $48 \cdot 2^{40}$ additions.

Fourth Step. At the fourth step of the partial sums attack, the adversary guesses the byte k_4, and computes $\oplus_{\{x \in \{0,1\}^8 : A_3[x]=1\}} S^{-1}(k_4 \oplus x)$, which is equal to the right hand side of (2), and checks whether it is equal to zero.

We cannot compute this XOR directly using a convolution, since in order to apply the FFT we need functions whose output is an integer and not an element of $GF(2^8)$. A basic solution, that was adopted by Todo and Aoki [30], is to compute the XOR in each bit separately. To this end, we define the functions $\chi^4_{k_0,k_1,k_2,k_3}, I^{4,j} : \{0,1\}^8 \to \{0,1\}$ for $j = 0, 1, \ldots, 7$ by

$$\chi^4_{k_0,k_1,k_2,k_3}(a_3) = 1(A_3[k_3][a_3]) \qquad \text{and} \qquad I^{4,j}(x) = [S^{-1}(x)]_j,$$

where $[S^{-1}(x)]_j$ denotes the j'th bit of $S^{-1}(x)$. We have

$$\begin{aligned}(\chi^4_{k_0,k_1,k_2,k_3} * I^{4,j})(k_4) &= \sum_{a_3 \in \{0,1\}^8} \chi^4_{k_0,k_1,k_2,k_3}(a_3) \cdot I^{4,j}(a_3 \oplus k_4) \\ &= \sum_{a_3 \in \{0,1\}^8} \mathbb{1}(A_3[k_3][a_3]) \cdot [S^{-1}(a_3 \oplus k_4)]_j.\end{aligned}$$

Therefore, the j'th bit of the XOR we would like to compute for the key guess $(k_0, k_1, k_2, k_3, k_4)$ is equal to

$$\left((\chi^4_{k_0,k_1,k_2,k_3} * I^{4,j})(k_4)\right) \bmod 2. \tag{7}$$

Algorithm 3. The following is the Algorithm for key recovery. The function $\mathbb{1}$ is the indicator function. All the blue colored steps are of complexity $2^{16} \times 2^{16}$ and can be replaced by a 3 Hadamard transformations of size 2^{16} with total complexity 3×2^{20}. The red colored step has complexity $2^8 \times 2^8$, which can be replaced by 3 Hadamard transformations of size 2^8 with total complexity 3×2^{11}.

Input: Array A of bits such that the j^{th} value of A denotes the parity of ciphertext j
Declare an empty 2D bit-array A_1 of size $2^{16} \times 2^{24}$; ▷ 2^{40} memory
for all a_1, c_2, c_3 **do**
  for all k_0, k_1 **do**
    $A_1[k_0, k_1][a_1, c_2, c_3] \leftarrow \bigoplus_{c_0, c_1} A[c_0, c_1, c_2, c_3] \cdot \mathbb{1}(S_0(c_0 \oplus k_0) \oplus S_1(c_1 \oplus k_1) = a_1)$
for all k_0, k_1 **do**
  Declare an empty 2D bit-array A_2 of size $2^8 \times 2^{16}$;
  for all c_3 **do**
    for all k_2, a_2 **do**
      $A_2[k_2][a_2, c_3] \leftarrow \bigoplus_{a_1, c_2} A_1[k_0, k_1][a_1, c_2, c_3] \cdot \mathbb{1}(a_1 \oplus S_2(c_2 \oplus k_2) = a_2)$
  for all k_2 **do**
    Declare an empty 2D bit-array A_3 of size $2^8 \times 2^8$;
    for all k_3, a_3 **do**
      $A_3[k_3][a_3] \leftarrow \bigoplus_{a_2, c_3} A_2[k_2][a_2, c_3] \cdot \mathbb{1}(a_2 \oplus S_3(c_3 \oplus k_3) = a_3)$
    for all k_3 **do**
      Declare an empty 1D byte-array A_4 of size 2^8;
      for all k_4 **do**
        $A_4[k_4] \leftarrow \bigoplus_{a_3} A_3[k_3][a_3] \cdot S^{-1}(a_3 \oplus k_4)$
      for all k_4 **do**
        if $A_4[k_4] \neq 0$ **then**
          k_0, k_1, k_2, k_3, k_4 is not a valid key candidate

Hence, we can check the XOR by initializing a list of 2^{40} binary indicators which corresponds to the possible values of $(k_0, k_1, k_2, k_3, k_4)$, computing the convolutions $\chi^4_{k_0,k_1,k_2,k_3} * I^{4,j}$ for $j = 0, 1, \ldots, 7$, and discarding all keys $(k_0, k_1, k_2, k_3, k_4)$ for which at least one of the results of (7) is not equal to zero modulo 2.

The complexity of this step is $2^{32} \cdot 8 \cdot (3 \cdot 8 \cdot 2^8) = 192 \cdot 2^{40}$ additions, which is slightly better than the complexity of the fourth step of the partial sums technique. As we shall show below, the complexity can be reduced significantly, by using a new method to pack several FFTs together, and exploiting enhancements from previous attacks based on the re-use of computations.

3.2 Packing Several FFTs Together by Embedding into $\mathbb{Z}$

We now show that the complexity of the basic attack can be significantly reduced by packing several convolution computations into a single convolution.

We assume that the attack is implemented using 64-bit operations, which is typical for a software implementation. For reference, the 6-round AES attack of Todo and Aoki requires 64-bit additions to avoid overflow.

Improving the Fourth Step of the Attack. Consider the fourth step of our basic attack described above. The step consists of computing the convolution of the function $\chi^4_{k_0,k_1,k_2,k_3}$ with the eight functions $I^{4,j}$ $(j = 0, 1, \ldots, 7)$. These eight convolutions can be replaced by a single computation of convolution.

Let s be a 'separation parameter' that will be determined below, and define a function $I^4 : \{0,1\}^8 \to \mathbb{Z}$ by $I^4(x) = \sum_{j=0}^{7} 2^{js}[S^{-1}(x)]_j$.

We claim that for an appropriate choice of s, the convolution $\chi^4_{k_0,k_1,k_2,k_3} * I^4$ allows recovering the value of the XOR in all 8 bits we are interested in, with a high probability. Indeed, we have

$$\begin{aligned}
(\chi^4_{k_0,k_1,k_2,k_3} * I^4)(k_4) &= \sum_{a_3 \in \{0,1\}^8} \chi^4_{k_0,k_1,k_2,k_3}(a_3) \cdot I^4(a_3 \oplus k_4) \\
&= \sum_{a_3 \in \{0,1\}^8} \mathbb{1}(A_3[k_3][a_3]) \cdot \sum_{j=0}^{7} 2^{sj}[S^{-1}(a_3 \oplus k_4)]_j \\
&= \sum_{j=0}^{7} 2^{sj} \sum_{a_3 \in \{0,1\}^8} \mathbb{1}(A_3[k_3][a_3]) \cdot [S^{-1}(a_3 \oplus k_4)]_j \\
&= \sum_{j=0}^{7} 2^{sj} (\chi^4_{k_0,k_1,k_2,k_3} * I^{4,j})(k_4),
\end{aligned}$$

where the penultimate equality uses the change of the order of summation.

Recall that for each value of k_4, we want to compute the eight parity bits $(\chi^4_{k_0,k_1,k_2,k_3} * I^{4,j}(k_4)) \bmod 2$. Let us reformulate our goal, for the sake of convenience. Denoting $b_j = \chi^4_{k_0,k_1,k_2,k_3} * I^{4,j}(k_4)$, we have $\chi^4_{k_0,k_1,k_2,k_3} * I^4(k_4) = \sum_{j=0}^{7} 2^{sj} b_j$. Thus, for non-negative integers $b_0, b_1, \ldots, b_7$, we are given $\sum_{j=0}^{7} 2^{sj} b_j$ and we want to compute from it the eight parity bits $(b_j) \bmod 2$.

Observe that if for all $0 \le j \le 7$, we have $b_j < 2^s$, then the multiplications by 2^{sj} separate the values b_j, and thus, we can simply read the values $(b_j) \bmod 2$ from $2^{sj} b_j$, as in this case,

$$\forall j : \left[\sum_{j=0}^{7} 2^{sj} b_j \right]_{sj} = [2^{sj} b_j]_{sj} = (b_j) \bmod 2.$$

How Large Should s be so that $b_j < 2^s$ Holds with a High Probability for All j's? Note that each b_j is the sum of 128 elements, which correspond to the 128 values of c_3 such that $[S^{-1}(c_3 \oplus k_4)]_j = 1$. Each such element is $\chi^4_{k_0,k_1,k_2,k_3}(c_3)$, which can be viewed as a randomly distributed indicator. Hence, b_j is distributed

like $Bin(128, 1/2)$. The expectation of such a variable is 64, and its standard deviation is $4\sqrt{2}$. This means that the values b_j are strongly concentrated around 64, and the probability $\Pr[b_j \geq 2^7]$ is extremely small. Therefore, by taking $s = 7$, we can derive the eight parity bits $(b_j) \bmod 2$ from the sum $\sum_{j=0}^{7} 2^{sj} b_j$, easily and with a very low error probability.

How Small Should s be in Order to Perform the Entire Computation with 64-bit Words? For the sake of efficiency, we compute the convolution using 64-bit word operations and disregard overflow beyond the 64'th bit. If s is too large, this may cause an error in the computation of the sum $\sum_{j=0}^{7} 2^{sj} b_j$, and consequently, in the computation of the parity bits $(b_j) \bmod 2$.

To overcome this, note that in the computation of a convolution of $f, g : \{0,1\}^n \to \mathbb{Z}$, all operations are additions and multiplications, except for division by 2^n at the last step. Hence, when we neglect overflow beyond the 64'th bit, this causes an additive error of $m \cdot 2^{64}$ for some $m \in \mathbb{Z}$ until the last step, and an additive error of $m \cdot 2^{64-n}$ at the final result. Assuming that $b_j < 2^s$ for all j, this error does not affect the parity bits as long as $7s < 64 - n$ (as the error affects only the top n bits of $\sum_{j=0}^{7} 2^{sj} b_j$).

In our case, $n = 8$ and hence, for all $s \leq 7$, the possible error does not affect the parity bits we compute.

Reducing s Even Further. Note that we can allow random errors in the convolution computations that do not correspond to the right subkey guess, as such random errors do not increase the probability of a wrong key guess to pass the filtering. Hence, we only have to make sure that for the right key, we obtain the correct value of the parity bits with a high probability.

As was explained above, the values b_j are concentrated around 64. Formally, by evaluating the cumulative distribution function of the binomial law, we have $\Pr[48 < b_j < 80] > 0.99$, and thus, $0 < b_j - 48 < 2^5$ with a very high probability. To make use of this concentration, we subtract from the value $\sum_{j=0}^{7} 2^{sj} b_j$ the integer $u = 48 \sum_{j=0}^{7} 2^{sj}$, to obtain

$$\sum_{j=0}^{7} 2^{sj} b_j - \sum_{j=0}^{7} 48 \cdot 2^{sj} = \sum_{j=0}^{7} (b_j - 48) 2^{sj}.$$

Since $0 < b_j - 48 < 2^5$, we can compute the parity bits $(b_j) \bmod 2$ also for $s = 6$ and for $s = 5$, with a very low error probability.

Summary of improving the fourth step. To summarize, the eight convolutions can be computed using a single convolution of functions over $\{0,1\}^8$. This reduces the complexity of this step to $2^{32} \cdot 3 \cdot 8 \cdot 2^8 = 24 \cdot 2^{40}$ operations.

Improving the Other Steps of the Attack. Once we acquired the ability to compute several convolutions in parallel, we can use it at the other steps of the attack as well. The idea is to pack the convolutions that correspond to several

subkey guesses into a single convolution. We exemplify this approach by showing how the first step of the attack can be improved; the improvement of the second and the third steps are similar.

Recall that at the first step of our attack, for any values $c_2, c_3 \in \{0,1\}^8$, we compute the parity of the convolution $(\chi^1_{c_2,c_3} * I^1_{a_1})(k_0, k_1)$, for all $k_0, k_1 \in \{0,1\}^8$. We may pack up to seven such computations in parallel. For example, in order to pack four computations, we write $c_2 = (c_2^h, c_2^l)$, where c_2^h denotes the two most significant bits of c_2 and is identified with an integer between 0 and 3, via the binary expansion. We define

$$\chi^1_{c_2^h,c_2^l,c_3}(c_0,c_1) = \chi(c_0,c_1,c_2,c_3), \text{ and } \bar{\chi}^1_{c_2^l,c_3} = \sum_{j=0}^{3} 2^{sj} \chi^1_{j,c_2^l,c_3}.$$

Then, for any $c_2^l \in \{0,1\}^6$, and $k_0, k_1, c_3 \in \{0,1\}^8$, we compute the convolution $(\bar{\chi}^1_{c_2^l,c_3} * I^1_{a_1})(k_0, k_1)$, and using the technique described above we derive from it the four parity bits $((\chi^1_{c_2,c_3} * I^1_{a_1})(k_0, k_1)) \bmod 2$ with $c_2 \in \{(0, c_2^l), \ldots, (3, c_2^l)\}$.

To see what is the maximal value of s we may take, note that each convolution value $b' = (\chi^1_{c_2,c_3} * I^1_{a_1})(k_0, k_1)$ is the sum of 256 elements, which correspond to the 256 values of (c_0, c_1) such that $S_0(c_0 \oplus k_0) \oplus S_1(c_1 \oplus k_1) = a_1$. Each such element can be viewed as a randomly distributed indicator. Hence, b' is distributed like $Bin(256, 1/2)$. When analyzing step 4, we could tolerate a low probability of errors for the right key, but in the first step, there are 2^{24} values of A_1 that are involved in the computation for the right key, and we want all of them to be correct. Therefore, we use $s \geq 7$, since $\Pr[64 < b' < 192] > 1 - 2^{-50}$. Hence, by subtracting $64 \cdot \sum_{j=0}^{3} 2^{js}$ from the convolution value $(\bar{\chi}^1_{c_2^l,c_3} * I^1_{a_1})(k_0, k_1)$, we can correctly compute the parity bits $\left((\chi^1_{c_2,c_3} * I^1_{a_1})(k_0, k_1)\right) \bmod 2$ with a very high probability for $s \geq 7$, and the 2^{24} relevant values are simultaneously correct with probability at least $1 - 2^{-26}$.

Unfortunately, with $s = 7$ we can only pack 7 parallel convolutions within 64-bit words. Indeed, at this step, the convolution is computed for functions over $\{0,1\}^{16}$ (instead of 8-bit functions in the fourth step), and thus, we need $7s < 64 - 16 = 48$ in order to pack 8 FFTs and avoid errors due to overflow. (We exemplified the idea of packing 4 parallel convolutions for the sake of convenience).

This reduces the complexity of the first step of the attack from $2^{24} \cdot 3 \cdot 16 \cdot 2^{16} = 48 \cdot 2^{40}$ to $48/7 \cdot 2^{40}$ addition operations. The complexity of the second step can be reduced similarly from $48 \cdot 2^{40}$ to $48/7 \cdot 2^{40}$. For the third step, we can actually use $s = 6$ and pack 8 parallel convolutions within a 64-bit word, because we only need 2^8 correct computations, and we have $\Pr[96 < b' < 160]^{256} > 0.98$; the complexity is reduced from $48 \cdot 2^{40}$ to $6 \cdot 2^{40}$.

Improving the Fourth Step Even Further. Finally, we can reduce the complexity of the fourth step even further by packing 12 FFTs in a 64-bit word with $s = 5$. This requires changing as described above the way we do the packing:

instead of packing 8 different $I^{4,j}$ with a fixed χ^4, we consider each function $I^{4,j}$ separately and pack a fixed $I^{4,j}$ with 12 χ^4 functions corresponding to different key guesses. This reduces the complexity of the fourth step from $24 \cdot 2^{40}$ to $16 \cdot 2^{40}$.

3.3 Enhancements and Other Variants of the Basic Technique

In this section, we present two enhancements that reduce the complexity of the attack, along with another variant of the technique that provides us with flexibility that will be useful in the application of our technique to other ciphers.

Precomputing Some of the FFT Computations. At each step of the attack, we perform three FFT computations. As was described in Sect. 2.4 regarding the FFT-based attack of Todo and Aoki, some of these computations do not depend on the guessed key material, and hence, they can be precomputed at the beginning of the attack, thus reducing the overall time complexity.

Specifically, the functions I^2, I^3, I^4, and $I^1_{a_1}$ (for all $a_1 \in \{0,1\}^8$) do not depend on any guessed subkey bits, and thus, their FFTs can be precomputed with overall complexity of about $2^8 \cdot 16 \cdot 2^{16} = 2^{28}$ addition operations, which is negligible compared to other steps of the attack. The results can be stored in lists that require about 2^{24} 64-bit words of memory.

The function $\chi^1_{c_2,c_3}$ does not depend on the value of a_1, and thus, its FFT can be computed once (for each value of (c_2, c_3)) and reused for all values of a_1. This reduces the time complexity of this FFT computation (in total, for all values of c_2, c_3) to $2^{16} \cdot 16 \cdot 2^{16} = 2^{36}$ additions, which is negligible compared to other steps of the attack. As we need to store in memory at each time only the result of the FFT that corresponds to a single value of c_2, c_3, the memory requirement of this step is 2^{16} 64-bit words of memory.

These precomputations reduce the time complexity of the first step (in which two FFTs can be precomputed) from $48/7 \cdot 2^{40}$ to $16/7 \cdot 2^{40}$ additions, the time complexity of the second, third, and fourth steps (in which one FFT can be precomputed) to $32/7 \cdot 2^{40}, 4 \cdot 2^{40}$, and $32/3 \cdot 2^{40}$ additions, respectively.

If the fourth step is implemented by packing 12 χ^4 functions together, as was described above, we can reduce its complexity further by precomputing the FFT of the function $\bar{\chi}^4$ which represents the 'packed' function and reusing it for computing convolutions with the eight functions $I^{4,j}$ $(j = 0, 1, \ldots, 7)$. This reduces the time complexity of the fourth step to $(16 + (16/8))/3 \cdot 2^{40} = 6 \cdot 2^{40}$ additions.

Therefore, the time complexity of examining a set of 2^{32} plaintexts is reduced to $2^{40} \cdot (16/7 + 32/7 + 4 + 6) \approx 16.9 \cdot 2^{40} \approx 2^{44.1}$ additions.

Lower Cost for Examining Additional Sets of Plaintexts. As was described in Sect. 2.3 regarding the partial sums attack, when we check the XOR of additional sets of 2^{32} values at a byte which we already checked for one set, the complexity of the check is reduced. Indeed, after the first set was checked, we expect that for each value of $(k^5_0, k^5_7, k^5_{10}, k^5_{13})$, only a few values of

$\bar{k}_0^4$ are not discarded. Hence, instead of performing the fourth step of the attack by computing a convolution, we can simply compute the sum directly for each of the remaining candidate subkeys. The average complexity of such a step is $2^{32} \cdot 1 \cdot 2^7 = 2^{39}$ S-box evaluations and the same number of XORs, which is equivalent to about $1 \cdot 2^{40}$ addition operations. Note that since the fourth step is the most time consuming step of our attack, this gain is more significant than the gain which the partial sums attack achieves in the same case.

After two sets were checked, we expect that for each value of (k_0^5, k_7^5, k_{10}^5), only a few values of $(k_{13}^5, \bar{k}_0^4)$ are not discarded. Hence, instead of performing the third and the fourth steps of the attack by computing convolutions, we can simply perform each of them for each of the remaining candidate subkeys. This reduces the complexity of the third step to 2^{40} additions and the complexity of the fourth step to 2^{32} additions.

Attack that Examines Six Sets of 2^{32} *Plaintexts.* By continuing the reasoning in the same manner, we see that the complexity of considering six sets of 2^{32} ciphertexts and computing the XOR of the values x_0^4 that correspond to them, is about

$$2^{40} \cdot \left(\left(\frac{16+32}{7} + 4 + 6 \right) + \left(\frac{16+32}{7} + 4 + 1 \right) + \left(\frac{16+32}{7} + 1 \right) + \left(\frac{16}{7} + 1 \right) + 1 \right)$$
$$\approx 40.8 \cdot 2^{40} \approx 2^{45.4} \text{ additions.}$$

Attack that Examines Two Sets of 2^{32} *Plaintexts.* If we consider two sets of 2^{32} ciphertexts and examine 4 different bytes (as was suggested by Tunstall [31] for the partial sums attack), then we may begin with checking the XOR of both sets at the byte x_0^4, which requires $2^{40}(16.9 + 11.9)$ additions as was described above. Then, we must move to another byte, and it seems that we have to pay a 'full price' again. However, note that after the first two filterings, for each value of (k_0^5, k_7^5, k_{10}^5) we are left with one value of k_{13}^5 on average. As these four subkey bytes are reused in the examination of the XOR in the byte x_5^4 (along with a different byte from $\bar{k}^4$), we can replace the third step by computing the sum directly for each remaining value of k_{13}^5 and replace the fourth step by computing the sum directly for each remaining value of $(k_{13}^5, \bar{k}_1^4)$. This reduces the complexity of each of these two steps to 2^{40} additions. When we examine the second set of 2^{32} ciphertexts at the byte x_5^4, the complexity of the fourth step can be further reduced to 2^{32} additions, since for any value of (k_0^5, k_7^5, k_{10}^5) we are left with one value of $(k_{13}^5, \bar{k}_1^4)$ on average.

Continuing in the same manner, we see that the complexity of considering two sets of 2^{32} ciphertexts and computing the XOR of the values $x_{0,5,10,15}^4$ that correspond to them, is about

$$2^{40} \cdot \Big(\left(\frac{16+32}{7} + 4 + 6 \right) + \left(\frac{16+32}{7} + 4 + 1 \right) + \left(\frac{16+32}{7} + 1 + 1 \right) +$$
$$\left(\frac{16+32}{7} + 1 \right) + \left(\frac{16}{7} + 1 \right) + \left(\frac{16}{7} + 1 \right) + 1 \Big) \approx 62.8 \cdot 2^{40} \approx 2^{46} \text{ additions.}$$

Attack that Examines One Set of 2^{32} Plaintexts. As was explained in Sect. 2.3, in this case we examine each byte with only a single set of ciphertexts, and thus, we do not obtain information that can be reused in other computations. Therefore, the complexity of our attack in this case is $16 \cdot 16.9 \cdot 2^{40} = 2^{48.1}$ addition operations, which is 16 times the complexity of checking a single set of ciphertexts (like in the partial sums and the Todo-Aoki attacks with only a single set of 2^{32} ciphertexts examined).

Alternative Way of Performing the First Step. Recall that at the first step we are given a list A of 2^{32} binary indices which correspond to (c_0, c_1, c_2, c_3) and our goal is to compute the 2^{24} entries of the list A_1 which corresponds to triples of the form (a_1, c_2, c_3) where $a_1 = S_0(c_0 \oplus k_0) \oplus S_1(c_1 \oplus k_1)$, for all values of (k_0, k_1). We may divide this step into two sub-steps as follows:

- *Step 1.1:* At this sub-step, we guess the subkey k_0 and update the list A into a list A_0 of 2^{32} binary indices that correspond to (a_0, c_1, c_2, c_3), where $a_0 = S_0(c_0 \oplus k_0)$. The complexity of this step is about $2^{32} \cdot 2^8 = 2^{40}$ S-box computations.
- *Step 1.2:* At this sub-step, performed for each guess of k_0, our goal is to replace the list A_0 with a list of size 2^{24} that corresponds to the values (a_1, c_2, c_3) where $a_1 = a_0 \oplus S_1(c_1 \oplus k_1)$, for each value of k_1. This task is exactly the same as the task handled at the second and third steps of our attack described above, and hence, it can be performed in exactly the same way. Specifically, the convolution we have to compute is

$$A_1[k_0, k_1][a_1, c_2, c_3] = \left((\bar{\chi}^1_{k_0,c_2,c_3} * \bar{I}^1)(a_1, k_1)\right) \bmod 2, \tag{8}$$

  where

$$\bar{\chi}^1_{k_0,c_2,c_3}(a_0, c_1) = \mathbb{1}(A_0(a_0, c_1, c_2, c_3) = 1), \text{ and } \bar{I}^1(x, y) = \mathbb{1}(x = S_1(y)).$$

  Like in the second step of our attack described above, we can precompute one FFT and perform the computation of 7 FFTs in parallel. Hence, the complexity of this sub-step is $32/7 \cdot 2^{40}$ additions.

The alternative version of the attack is presented in Algorithm 4.

Formally, the complexity of the alternative way is higher than the complexity of the original way of performing this step described above—$39/7 \cdot 2^{40}$ additions instead of $16/7 \cdot 2^{40}$ additions. As a result, the complexity of the attack with two sets of 2^{32} plaintexts becomes about $82.5 \cdot 2^{40} \approx 2^{46.4}$ additions (which is the complexity we mention in the introduction). However, this alternative has several advantages:

1. *Lower memory complexity.* In the attack described above, the most memory-consuming part is the first step which requires a list of 2^{40} bit entries. Thus, its memory complexity is about 2^{33} 128-bit blocks.
   The alternative way reduces the memory complexity of the first step to 2^{32}

Algorithm 4. Low-memory version of the attack.

Input: Array A of bits such that the j^{th} value of A denotes the parity of ciphertext j
for all k_0 **do**
  Declare an empty 1D bit-array A_0 of size 2^{32}; ▷ 2^{32} memory
  for all c_0, c_1, c_2, c_3 **do**
    $a_0 \leftarrow S_0(c_0 \oplus k_0)$
    $A_0[a_0, c_1, c_2, c_3] \leftarrow A[c_0, c_1, c_2, c_3]$
  Declare an empty 2D bit-array A_1 of size $2^8 \times 2^{24}$; ▷ 2^{32} memory
  for all c_2, c_3 **do**
    for all k_1, a_1 **do**
      $A_1[k_1][a_1, c_2, c_3] \leftarrow \bigoplus_{a_0, c_1} A_0[a_0, c_1, c_2, c_3] \cdot \mathbb{1}(a_0 \oplus S_1(c_1 \oplus k_1) = a_1)$
  for all k_1 **do**
    Declare an empty 2D bit-array A_2 of size $2^8 \times 2^{16}$;
    for all c_3 **do**
      for all k_2, a_2 **do**
        $A_2[k_2][a_2, c_3] \leftarrow \bigoplus_{a_1, c_2} A_1[k_1][a_1, c_2, c_3] \cdot \mathbb{1}(a_1 \oplus S_2(c_2 \oplus k_2) = a_2)$
    for all k_2 **do**
      Declare an empty 2D bit-array A_3 of size $2^8 \times 2^8$;
      for all k_3, a_3 **do**
        $A_3[k_3][a_3] \leftarrow \bigoplus_{a_2, c_3} A_2[k_2][a_2, c_3] \cdot \mathbb{1}(a_2 \oplus S_3(c_3 \oplus k_3) = a_3)$
      for all k_3 **do**
        Declare an empty 1D byte-array A_4 of size 2^8;
        for all k_4 **do**
          $A_4[k_4] \leftarrow \bigoplus_{a_3} A_3[k_3][a_3] \cdot S^{-1}(a_3 \oplus k_4)$
        for all k_4 **do**
          if $A_4[k_4] \neq 0$ **then**
            k_0, k_1, k_2, k_3, k_4 is not a valid key candidate

bits. We observe that all other steps of the attack can be performed with at most 2^{34} bits of memory. Indeed, all ciphertexts can be transformed immediately into entries of the table A whose size is 2^{32} bits. The table A_0 (which should be stored for one value of k_0 at a time) requires 2^{32} bits. The subsequent tables used in the attack are smaller, and the arrays used in the FFTs are also smaller (as all FFTs are performed on 16-bit or 8-bit functions). By checking two sets of 2^{32} plaintexts in parallel, we reduce the number of remaining keys after examining the byte x_0^4 to 2^{24}, and then the storage of these keys requires less than 2^{30} bits of memory. Therefore, the total memory complexity of the attack is reduced to about $2 \cdot 2^{32} + 2^{32} < 2^{34}$ bits, i.e., 2^{27} 128-bit blocks.

2. *Lower average-case time complexity.* While it is common to measure the complexity of attacks using the worst-case scenario (e.g., the complexity of

exhaustive search over an n-bit key is computed as 2^n, although on average, the attack finds the key after 2^{n-1} trials), the average-case complexity has clear practical significance. In the partial sums attack and in the Todo-Aoki attack, the average-case time complexity is half of the worst-case complexity, since the attack is applied for 2^8 'external' guesses of a subkey, and the right key is expected to be found after trying half of these subkeys. In the original version of our attack, since the last step is performed for all keys in parallel, our average-case complexity is no better than the worst-case complexity, and so, we lose a factor of 2. In the alternative way described here, the attack is performed for each guess of the subkey k_0^5, and hence, we regain the factor 2 loss in the average-case complexity.

3. *Practical effect on the time complexity.* The lower memory complexity of the alternative variant of the attack is expected to have an effect on the time complexity as well. Indeed, our experiments show that the memory accesses to the 2^{40}-bit sized array slow down our attack considerably. As the alternative variant requires only 2^{34} bits of memory, it may be even faster in practice than the original variant.

The alternative way of performing the first step is also used in our improved attack on the full MISTY1 [24] presented in the full version of this paper [19].

3.4 Our Technique vs. Partial Sums and the Todo-Aoki Technique

In this section, we present a comparison between our new technique and the partial sums technique and the Todo-Aoki FFT-based technique. First, we discuss the case of 6-round AES, and then we discuss applications to general ciphers.

The Case of 6-Round AES. Here, we considered three attacks:

1. *Attack with 6 structures of* 2^{32} *chosen plaintexts.* The partial sums attack requires $2^{51.3}$ S-box computations, the Todo-Aoki attack requires $2^{50.8}$ additions, and our attack requires $2^{45.4}$ additions. Hence, our attack is at least 32 times faster than both previous attacks. In the experiments presented in Sect. 3.5, the advantage of our attack was even bigger.
2. *Attack with 2 structures of* 2^{32} *chosen plaintexts.* The partial sums attack requires $2^{51.7}$ S-box computations, the Todo-Aoki attack requires $2^{51.2}$ additions, and our attack requires 2^{46} additions. Hence, our attack is at least 32 times faster than both previous attacks.
3. *Attack with 1 structure of* 2^{32} *chosen plaintexts.* The partial sums attack requires 2^{54} S-box computations, the Todo-Aoki attack requires 2^{53} additions, and our attack requires $2^{48.1}$ additions. Hence, our attack is almost 32 times faster than both previous attacks.

General Comparison. The speedup of our technique over the partial sums technique stems from two advantages: First, we replace key guessing steps with

computation of convolutions. Second, we may pack the computation of several convolutions in a single convolution computation. The effect of the first advantage depends on the number of subkey bits guessed at the most time consuming steps of the attack: For a 4-bit subkey guess our gain is negligible, for an 8-bit key guess we get a speedup by a factor of more than 10 (without using packing), and for a 32-bit key guess our speedup factor may be larger than 2^{25} as demonstrated in our attack on CLEFIA [22] presented in the full version of this paper [19]. The effect of the second advantage is also dependent on the number of guessed subkey bits (since it determines the size of the functions whose convolution we have to compute, which in turn affects the number of convolutions we may pack together). Usually, between 4 and 8 convolutions can be packed together, which leads to a speedup by a factor of at least 4. Interestingly, when the number of guessed subkey bits is small (e.g., 4 bits), more convolutions can be packed together, and hence, a stronger effect of the second advantage compensates for a weaker effect of the first advantage.

The speedup of our technique over the Todo-Aoki technique stems from two advantages: First, our attack provides us with more flexibility, meaning that instead of replacing the whole attack by a single FFT-based step, we can consider each step (or a few steps) of the partial sums procedure separately and decide whether it will be better to perform it with key guessing or with an FFT-based technique. Second, we may pack the computation of several convolutions in a single convolution computation. The first advantage allows us to make use of partial knowledge of the subkey. A particular setting in which this advantage plays a role is analysis of additional plaintext sets after one set was used to obtain some key filtering. While our technique and the partial sums technique can make use of this partial knowledge, the Todo-Aoki technique must repeat the entire procedure. In the case of 6-round AES, this makes our attack 6 times faster than the Todo-Aoki attack without using packing. A more complete comparison between our method and the Todo-Aoki technique is available in the full version of this paper [19].

The second advantage provides a speedup by a factor of at least 4, as was described above. Yet another advantage that is worth mentioning is that while the Todo-Aoki technique applies the FFT to functions in high dimensions (e.g., dimension 72 in the Todo-Aoki attack on 12-round CLEFIA-128 presented in [30]), our technique applies the FFT to functions of a significantly lower dimension (e.g., dimension 16 in our improved attack on 12-round CLEFIA-128 presented in Appendix B of the full version of this paper [19]). Computation of the FFT in high dimensions is quite cumbersome from the practical point of view, and hence, avoiding this is a practical advantage of our technique. Moreover, higher dimension FFTs require additions with more precision; without using packing the Todo-Aoki attack on 6-round AES requires 64-bit additions while our attack can use 32-bit additions.

Two advantages of the partial sums technique and the Todo-Aoki technique over our technique are a somewhat lower memory complexity (about 2^{27} 128-bit blocks for partial sums and about 2^{31} 128-bit blocks for Todo-Aoki) and

the fact that on average, the attack finds the right key after trying half of the possible keys while our attack must try all keys. However, both advantages can be countered by implementing the first step of our attack in the alternative way presented in Sect. 3.3, which makes the memory complexity equal to that of the partial sums attack and regains the 'lower average-case complexity', as was explained in Sect. 3.3.

3.5 Experimental Verification of Our Attack on 6-Round AES

We have experimentally verified our attack on Amazon's AWS infrastructure. For comparison, we also implemented the partial sums attack of [21] and the Todo-Aoki attack [30]. All implementations in C are publicly available at the following link:

https://github.com/ShibamCrS/Partial_Sums_Meet_FFT.

We note that the FFT implementations were based on the "Fast Fast Hadamard Transform" library [3].

The AWS Instances Used in the Experiment. For each attack we had to pick the most optimal AWS instance, depending on the computational and memory requirements.

The partial sums attack is quite easy to parallelize, and its memory requirement is low. (Specifically, the memory requirement is 2^{34} bits, or 16GB, as was shown above. Furthermore, only an 2^{32}-bit list that stores the parities of (c_0, c_1, c_2, c_3) combinations should be stored in a memory readable by all threads). As a result, we took the Intel-based instance (that has the AES-NI instruction set) with the maximal number of cores per US$. At the time the experiment was performed (January, 2023) this was the m6i.32xlarge instance.[4]

For our attack (in its original variant) and for the Todo-Aoki attack, we needed instances that support a larger amount of memory. The optimal choice for our attack was the same instance as the one for the partial sums attack—the m6i.32xlarge instance. For the Todo-Aoki attack, we needed 64 GB of RAM for each thread of the attack. Hence, the optimal instance we found was the r6i.32xlarge instance.[5] We note that in the Todo-Aoki attack, we do not exploit all the vCPUs, but we do exploit the whole memory space (of 1 TB of RAM).

Experimental Results. The partial sums attack took 4859 minutes to complete, and its cost was 497 US$ (we used the US-east-2 region (Ohio) which offered the cheapest cost-per-hour for a Linux machine of 6.144 US$, before VAT). The Todo-Aoki approach took 3120 minutes to complete, and its cost was 418 US$ (at 8.064 US$ per hour). We note that due to the costs of these

[4] The m6i.32xlarge instance has 128 Intel-based vCPUs and 512GB of RAM.
[5] The r6i.32xlarge instance has 128 Intel-based vCPUs and 1024 GB of RAM.

attacks, they were run only once, but none of those attacks (nor our attack) is expected to show high variance in the running time.

To evaluate the running time of our attack, we ran Algorithm 3 and Algorithm 4 ten times each. In both algorithms, we used only 4 FFTs packed in parallel at each of Steps 1,2,3 and 8 FFTs packed in parallel at Step 4, for ease of implementation. The average running time of Algorithm 3 is 90 minutes, and its average cost is 9.21 US$. The average running time of Algorithm 4 is 48 minutes and its cost is 5 US$. Hence, in the experiment our attack was 83-times cheaper and 65 times faster than both partial sums and Todo-Aoki's attacks.

4 Improved Attack on Kuznyechik

The flexibility of our techniques improves attacks against various other ciphers that use the partial sums technique. In this section, we demonstrate this by presenting an attack on 7-round Kuznyechik, which improves over the multiset-algebraic attack on the cipher presented in [12] by a factor of more than 80. In the supplementary material of the full paper [19], we present improved attacks on the full MISTY1, and variants of CLEFIA-128 with 11 and 12 rounds. Our attacks on Kuznyechik and MISTY1 are the best known attacks on these ciphers.

4.1 The Structure of Kuznyechik

The block cipher Kuznyechik [18] is the current encryption standard of the Russian Federation. It is an SPN operating on a 128-bit state organized as a 4×4 array of 8-bit words. The key length is 256 bits, and the encryption process is composed of 9 rounds. Each round of Kuznyechik is composed of three operations:

Substitution. Apply an 8-bit S-box independently to every byte of the state;
Linear Transformation. Multiply the state by an invertible 16-by-16 matrix M over $GF(2^8)$;
Key Addition. XOR a 128-bit round key computed from the secret key to the state.

An additional key addition operation is applied before the first round. As properties of the key schedule of Kuznyechik are not used in this paper, we omit its description and refer the reader to [18].

4.2 The Multiset-Algebraic Attack of Biryukov et al.

In [12], Biryukov et al. presented an algebraic attack on up to 7 rounds of Kuznyechik. The attack is based on the following observation:

Lemma 3. *Consider the encryption by 4-round Kuznyechik of a set of* 2^{127} *distinct plaintexts,* $P^0, P^1, \ldots, P^{2^{127}-1}$*, which form a subspace of degree 127 of* $\{0,1\}^{128}$*. Then the corresponding ciphertexts satisfy* $\bigoplus_{i=0}^{2^{127}-1} C^i = 0$.

The attack uses Lemma 3 in the same way as the Square attack on AES uses Lemma 1. The adversary asks for the encryption of the entire codebook of 2^{128} plaintexts. Then he guesses a single byte of the whitening subkey and for each guess, he finds a set of 2^7 values in that byte such that the corresponding values after the substitution operation form a 7-dimensional subspace of $\{0,1\}^8$. By taking these values along with all 2^{120} possible values in the other 15 bytes, the adversary obtains a set of 2^{127} plaintexts, whose corresponding intermediate values after one round satisfy the assumption of Lemma 3.

By the lemma, the XOR of the corresponding values at the end of the 5'th round is zero. In order to check this, the adversary guesses some subkey bytes in the last two rounds and partially decrypts the ciphertexts to compute the XOR in a single byte at the end of the 5'th round. The situation is similar to the AES, with the 'only' difference that since the linear transformation is a 16-by-16 matrix (and not a 4-by-4), one has to guess all 16 bytes of the last round subkey. The adversary guesses the last round subkey and one byte of the equivalent subkey of the penultimate round, partially decrypts the ciphertexts, and checks whether the values XOR to zero. Biryukov et al. suggested to significantly speed up this procedure using partial sums. Borrowing the notation from Sect. 2.3, the value of the byte in which the XOR should be computed can be written as:

$$\begin{aligned} x_0^5 = S^{-1}(\bar{k}_0^5 \oplus e_0 \cdot S^{-1}(C_0 \oplus k_0^6) \oplus e_1 \cdot S^{-1}(C_1 \oplus k_1^6) \oplus \dots \\ \oplus e_{14} \cdot S^{-1}(C_{14} \oplus k_{14}^6) \oplus e_{15} \cdot S^{-1}(C_{15} \oplus k_{15}^6)), \end{aligned} \tag{9}$$

where the constants $e_0, e_1, \dots, e_{15}$ are obtained from the matrix M^{-1} and the multiplication is defined over $GF(2^8)$. In the attack of Biryukov et al., the sum in the right hand side of (9) is computed using 16 steps of partial sums, where we begin with a list of 2^{128} binary indices which indicate the parity of occurrence of each ciphertext value, and at each step, another subkey byte k_i^6 is guessed and the size of the list is reduced by a factor of 2^8. Like in the partial sums attack on the AES, the two outstanding steps are the first step in which two subkeys are guessed and the list is squeezed to a list of size 2^{120}, and the last step in which the XOR of 2^8 values is computed under the guess of 17 subkey bytes.

The complexity of each step is 2^{144} S-box computations, and hence, the complexity of the entire procedure is 2^{148} S-box computations. Since the procedure provides only an 8-bit filtering, the adversary has to repeat it for each of the 16 bytes (and for each guess of the subkey byte at the first round). Therefore, the total time complexity of the attack is $2^8 \cdot 16 \cdot 2^{148} = 2^{160}$ S-box computations, which are equivalent (according to [12]) to $2^{154.5}$ encryptions.

The authors of [12] present also an attack on 6-round Kuznyechik. In this attack, they use the fact that for 3-round Kuznyechik, taking a vector space of degree 120 of plaintexts (instead of degree 127 above) is sufficient for guaranteeing that the ciphertexts XOR to zero. Hence, in order to attack 6-round Kuznyechik, an adversary can ask for the encryption of 2^{120} plaintexts which are equal in a single byte and assume all possible values in the other bytes. The corresponding intermediate values after one round form a vector space of degree 120, and hence, the corresponding intermediate values after 4 rounds XOR to

zero. This allows applying the same attack like above, where the overall complexity is reduced by a factor of 2^8 since there is no need to guess a subkey byte at the first round. Hence, the overall data complexity is 2^{120} chosen plaintexts and the time complexity is $2^{146.5}$ encryptions.

The attacks of [12] are the best known attacks on reduced-round Kuznyechik.

4.3 Improvement Using Our Technique

Just like for AES, we can replace each step of the partial sums procedure performed in [12] by computing a convolution. We can compute several convolutions in parallel by embedding into $\mathbb{Z}$ as well as precompute two FFTs required for the first step and one FFT required for each subsequent step. However, we can only compute 6 FFTs in parallel rather than 7, as we need 2^{120} values to be correct in the first step. This requires $s \geq 8$ and cannot accommodate 7 parallel FFTs; instead we use 6 parallel FFTs with $s = 9$ which guarantees no overflow. The complexity of the first step is reduced to $2^{120} \cdot 16 \cdot 2^{16}/6 = 8/3 \cdot 2^{136}$ additions and the complexity of the subsequent steps is reduced to $2^{120} \cdot 2 \cdot 16 \cdot 2^{16}/6 = 16/3 \cdot 2^{136}$ additions. At the last step (which computes the XOR of the values) we have to compute FFTs for the 8 bits of the SBox individually, but we use FFTs on 8-bit functions (instead of 16-bit ones), we can pack 8 computations in parallel, and we can precompute an additional FFT and reuse it for the computations of the eight bits. Hence, its amortized complexity is $2^{128} \cdot (1 + (1/8)) \cdot 8 \cdot 2^8 = 9 \cdot 2^{136}$ additions. We conclude that the analysis of a single set of 2^{127} ciphertexts, with a given guess of the whitening key, takes $(8/3 + 14 \cdot 16/3 + 9)2^{136} = 259/3 \cdot 2^{136} = 2^{142.4}$ additions.

Instead of examining the other 15 bytes using the same set of 2^{127} ciphertexts, we may construct additional sets of 2^{127} ciphertexts by taking other 127-dimensional subspaces at the end of the first round (which is possible since we ask for the encryption of the entire codebook and guess a subkey byte at the first round) and examining their XOR at the same byte at the end of the 5'th round. Like in the case of AES, when we examine the XOR at the same byte for a second set of ciphertexts, the complexity of the last step becomes negligible (as it is performed only for a few possible values of the subkey). When a third set of ciphertexts is examined, the two last steps become negligible, etc. By using seven sets of 2^{127} ciphertexts and examining each of them in three bytes, the complexity of the attack becomes

$$\begin{aligned} &2^8 \cdot 2^{136} \cdot 1/3 \cdot \big((259 + 232 + 216 + 200 + 184 + 168 + 152) + \\ &+ (136 + 136 + 120 + 104 + 88 + 72 + 56) + (40 + 40 + 24 + 8)\big) \\ &= 2^{144} \cdot 745 = 2^{153.5} \end{aligned}$$

additions, which are equivalent to about 2^{148} encryptions – a speedup by a factor of more than 80 compared to the attack of [12].

The attack on 6-round Kuznyechik can be improved similarly. The only difference is that we cannot use additional sets of plaintexts without increasing

the data complexity. Hence, for the same data complexity, the time complexity is reduced to $2^{146.4}$ additions, which are equivalent to $2^{140.9}$ encryptions – a speedup by a factor of more than 40.

5 Summary

In this paper we showed that the partial sums technique of Ferguson et al. [21] and the FFT-based technique of Todo and Aoki [30] can be combined into a new technique that allows enjoying *the best of the two worlds*. The combination improves over the best previously known attacks on 6-round AES by a factor of more than 32, as we verified experimentally.

Furthermore, the new technique allows improving other attacks—most notably, we improve the best known attack against Kuznyechik [18] by a factor of more than 80. Our method also yields the best known attack against the full MISTY1 [24] where we improve previous best result by a factor of 6, and improve the partial sums attacks against reduced-round CLEFIA [22] by varying factors (including a huge factor of 2^{30}, on 12-round CLEFIA-128) as shown in the full version of this paper [19]. We expect that our new technique will be used to improve other cryptanalytic attacks, and will (again) highlight the strength and potential of FFT-based techniques in improving cryptanalytic attacks.

Acknowledgements. The research was conducted in the framework of the workshop 'New directions in the cryptanalysis of AES', supported by the European Research Council under the ERC starting grant agreement n. 757731 (LightCrypt). The authors thank all the participants of the workshop for valuable discussions and suggestions.

The first, second and sixth authors were supported in part by the Center for Cyber, Law, and Policy in conjunction with the Israel National Cyber Directorate in the Prime Minister's Office and by the Israeli Science Foundation through grants No. 880/18 and 3380/19. The third and the fifth authors were supported by the European Research Council under the ERC starting grant agreement n. 757731 (LightCrypt) and by the BIU Center for Research in Applied Cryptography and Cyber Security in conjunction with the Israel National Cyber Bureau in the Prime Minister's Office. The fourth author is partially supported by ANR grants ANR-20-CE48-001 and ANR-22-PECY-0010.

References

1. Advanced Encryption Standard (AES). National Institute of Standards and Technology, NIST FIPS PUB 197, U.S. Department of Commerce, November 2001
2. Aldà, F., Aragona, R., Nicolodi, L., Sala, M.: Implementation and improvement of the partial sum attack on 6-round AES. In: Baldi, M., Tomasin, S. (eds.) Physical and Data-Link Security Techniques for Future Communication Systems. LNEE, vol. 358, pp. 181–195. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-23609-4_12
3. Andoni, A., Indyk, P., Laarhoven, T., Razenshteyn, I., Schmidt, L.: Fast Fast Hadamard Transform. https://github.com/FALCONN-LIB/FFHT

4. Ankele, R., Dobraunig, C., Guo, J., Lambooij, E., Leander, G., Todo, Y.: Zero-correlation attacks on tweakable block ciphers with linear TWEAKEY expansion. Cryptology ePrint Archive, Report 2019/185 (2019). https://eprint.iacr.org/2019/185
5. Bao, Z., Guo, C., Guo, J., Song, L.: TNT: how to tweak a block cipher. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020, Part II. LNCS, vol. 12106, pp. 641–673. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45724-2_22
6. Bar-On, A., Dinur, I., Dunkelman, O., Lallemand, V., Keller, N., Tsaban, B.: Cryptanalysis of SP networks with partial non-linear layers. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9056, pp. 315–342. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46800-5_13
7. Bar-On, A., Dunkelman, O., Keller, N., Ronen, E., Shamir, A.: Improved key recovery attacks on reduced-round AES with practical data and memory complexities. J. Cryptol. **33**(3), 1003–1043 (2020)
8. Bar-On, A., Keller, N.: A 2^{70} attack on the full MISTY1. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, Part I. LNCS, vol. 9814, pp. 435–456. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53018-4_16
9. Bariant, A., Leurent, G.: Truncated boomerang attacks and application to AES-based ciphers. In: Hazay, C., Stam, M. (eds.) EUROCRYPT 2023. LNCS, vol. 14007, pp. 3–35. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-30634-1_1
10. Beierle, C., Leander, G., Todo, Y.: Improved differential-linear attacks with applications to ARX ciphers. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part III. LNCS, vol. 12172, pp. 329–358. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-56877-1_12
11. Biryukov, A.: The boomerang attack on 5 and 6-round reduced AES. In: Dobbertin, H., Rijmen, V., Sowa, A. (eds.) AES 2004. LNCS, vol. 3373, pp. 11–15. Springer, Heidelberg (2005). https://doi.org/10.1007/11506447_2
12. Biryukov, A., Khovratovich, D., Perrin, L.: Multiset-algebraic cryptanalysis of reduced Kuznyechik, Khazad, and secret SPNs. IACR Trans. Symm. Cryptol. **2016**(2), 226–247 (2016). https://tosc.iacr.org/index.php/ToSC/article/view/572
13. Bogdanov, A., Geng, H., Wang, M., Wen, L., Collard, B.: Zero-correlation linear cryptanalysis with FFT and improved attacks on ISO standards Camellia and CLEFIA. In: Lange, T., Lauter, K., Lisoněk, P. (eds.) SAC 2013. LNCS, vol. 8282, pp. 306–323. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-43414-7_16
14. Collard, B., Standaert, F.-X., Quisquater, J.-J.: Improving the time complexity of Matsui's linear cryptanalysis. In: Nam, K.-H., Rhee, G. (eds.) ICISC 2007. LNCS, vol. 4817, pp. 77–88. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-76788-6_7
15. Cui, J., Hu, K., Wang, Q., Wang, M.: Integral attacks on Pyjamask-96 and round-reduced Pyjamask-128. In: Galbraith, S.D. (ed.) CT-RSA 2022. LNCS, vol. 13161, pp. 223–246. Springer, Cham (2022). https://doi.org/10.1007/978-3-030-95312-6_10
16. Daemen, J., Knudsen, L., Rijmen, V.: The block cipher Square. In: Biham, E. (ed.) FSE 1997. LNCS, vol. 1267, pp. 149–165. Springer, Heidelberg (1997). https://doi.org/10.1007/BFb0052343
17. Demirbaş, F., Kara, O.: Integral characteristics by keyspace partitioning. Des. Codes Crypt. **90**(2), 443–472 (2022)
18. Dolmatov, V., e.: Gost r 34.12-2015: Block cipher "Kuznyechik" (2016). https://www.rfc-editor.org/rfc/rfc7801.html

19. Dunkelman, O., Ghosh, S., Keller, N., Leurent, G., Marmor, A., Mollimard, V.: Partial sums meet FFT: Improved attack on 6-round AES. Cryptology ePrint Archive, Paper 2023/1659 (2023). https://eprint.iacr.org/2023/1659
20. Dunkelman, O., Keller, N., Ronen, E., Shamir, A.: The retracing boomerang attack. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020, Part I. LNCS, vol. 12105, pp. 280–309. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45721-1_11
21. Ferguson, N., Kelsey, J., Lucks, S., Schneier, B., Stay, M., Wagner, D.: Improved cryptanalysis of Rijndael. In: Goos, G., Hartmanis, J., van Leeuwen, J., Schneier, B. (eds.) FSE 2000. LNCS, vol. 1978, pp. 213–230. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44706-7_15
22. Katagi, M.: The 128-bit blockcipher CLEFIA (2011). https://www.rfc-editor.org/rfc/rfc6114
23. Li, Y., Wu, W., Zhang, L.: Improved integral attacks on reduced-round CLEFIA block cipher. In: Jung, S., Yung, M. (eds.) WISA 2011. LNCS, vol. 7115, pp. 28–39. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-27890-7_3
24. Matsui, M.: New block encryption algorithm MISTY. In: Biham, E. (ed.) FSE 1997. LNCS, vol. 1267, pp. 54–68. Springer, Heidelberg (1997). https://doi.org/10.1007/BFb0052334
25. Nguyen, P.H., Wei, L., Wang, H., Ling, S.: On multidimensional linear cryptanalysis. In: Steinfeld, R., Hawkes, P. (eds.) ACISP 2010. LNCS, vol. 6168, pp. 37–52. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14081-5_3
26. Nguyen, P.H., Wu, H., Wang, H.: Improving the algorithm 2 in multidimensional linear cryptanalysis. In: Parampalli, U., Hawkes, P. (eds.) ACISP 2011. LNCS, vol. 6812, pp. 61–74. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22497-3_5
27. Rahman, M., Saha, D., Paul, G.: Boomeyong: embedding yoyo within boomerang and its applications to key recovery attacks on AES and Pholkos. IACR Trans. Symmetric Cryptol. **2021**(3), 137–169 (2021)
28. Sasaki, Y., Wang, L.: Meet-in-the-middle technique for integral attacks against feistel ciphers. In: Knudsen, L.R., Wu, H. (eds.) SAC 2012. LNCS, vol. 7707, pp. 234–251. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-35999-6_16
29. Todo, Y.: FFT-based key recovery for the integral attack. Cryptology ePrint Archive, Report 2014/187 (2014). https://eprint.iacr.org/2014/187
30. Todo, Y., Aoki, K.: FFT key recovery for integral attack. In: Gritzalis, D., Kiayias, A., Askoxylakis, I. (eds.) CANS 2014. LNCS, vol. 8813, pp. 64–81. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-12280-9_5
31. Tunstall, M.: Improved "partial sums"-based square attack on AES. In: Proceedings of the International Conference on Security and Cryptography - SECRYPT, (ICETE 2012), pp. 25–34. INSTICC, SciTePress (2012)
32. Yi, W., Chen, S., Wei, K.: Zero-correlation linear cryptanalysis of reduced round ARIA with partial-sum and FFT. arXiv preprint arXiv:1406.3240 (2014)