

KIFA: Kernel-based Integral Fault Analysis on GIFT-Family Ciphers

Hanbeom Shin¹, Yunjae Hwang¹, Donggeun Kwon², Byoungjin Seok³,
Deukjo Hong⁴, Jaechul Sung⁵, Seokhie Hong⁶, and Dongjae Lee⁷

¹ Korea University, Seoul, Republic of Korea

`newonetiger@korea.ac.kr`, `hyj019@korea.ac.kr`

² Kunsan National University, Gunsan, Republic of Korea

`dgkwon@kunsan.ac.kr`

³ Hansung University, Seoul, Republic of Korea

`bjseok@hansung.ac.kr`

⁴ Jeonbuk National University, Jeonju, Republic of Korea

`deukjo.hong@jbnu.ac.kr`

⁵ University of Seoul, Seoul, Republic of Korea

`jcsung@uos.ac.kr`

⁶ SmartM2M, Seoul, Republic of Korea

`shhong@smartm2m.co.kr`

⁷ Kangwon National University, Chuncheon, Republic of Korea

`dongjae.lee@kangwon.ac.kr`

Abstract. We present KIFA (Kernel-based Integral Fault Analysis), an integral fault attack framework that mines its balanced properties with the kernel-based miner of Hwang et al. in place of Mixed Integer Linear Programming (MILP). The miner handles only small input structures fixed in advance, and a fault model provides exactly such structures. For **DEFAULT**, we present attacks at fault penetration (FP) 9 under a random-position nibble fault and FP 10 under a random-position byte fault, beyond the prior best integral fault attack at FP 8. They leave a single master-key candidate under the simple key schedule and reduce the key space to 2^{64} under the rotating key schedule. For **GIFT-128**, we present the first integral fault attacks, at FP 7 and 8, against the prior best of 5. For **GIFT-64**, we extend the prior best FP 6 to 7 with a byte fault, leaving 2^{32} master-key candidates. For **BAKSHEESH**, we extend the prior best FP 7 to 8, also when the fault position is uncontrolled.

Keywords: Fault Attack · Division Property-Based Fault Analysis · Kernel-Based Balanced Property · **DEFAULT** · **GIFT** · **BAKSHEESH**

1 Introduction

Fault attacks recover the secret key by analyzing the faulty outputs that a cryptographic device produces after a fault is injected during execution, for example by a laser pulse [1,21], electromagnetic interference [19], a voltage or clock glitch [10,17], supply-voltage underfeeding [7], or a memory upset [15]. The resulting fault is typically modeled either as a single bit-flip at a known position

or as a random nibble or random byte fault, where the injected value can take any of 2^4 or 2^8 values, respectively. These random models are realistic since they require neither control of the faulty value nor precise injection timing; when the fault position is uncontrolled as well, the model is a *random-position* fault. Integral fault attacks operate in these models. Which of the two widths applies is set by the implementation rather than chosen by the attacker.

Two factors determine the impact of a fault attack on practical deployments, the *fault penetration* (FP), defined as the number of cipher rounds covered by injected faults, and the number of faults consumed by the attack. Of the two, the FP is more critical against redundancy-based countermeasures, which detect injected faults by performing a redundant computation over the last few rounds and rejecting any output whose redundant value differs from the original. Because the redundant computation incurs substantial area and power overhead, deployments restrict it to the last few rounds in which every prior fault attack is known to operate [17]. A new attack at a deeper round forces the protected window to expand. On GIFT-128, for example, our 8-round byte attack widens the redundant window from the 5/40 rounds of the prior best attack to 8/40 and increases the implementation cost accordingly.

Fault attacks on block ciphers began with differential fault analysis (DFA) [9], which compares correct and faulty ciphertexts of the same plaintext. In the integral fault attack line that this paper follows, DiFA (Division Property-Based Fault Analysis) [18] first combined the division property [23,24] with fault attacks, recovering the GIFT-64 master key at FP 6, counted from its first set of faults, with a second set injected two rounds earlier, and DIFA-Rent [12] extended it by one round using the linear structure of the S-box, recovering the BAKSHEESH master key uniquely at FP 7 and reducing the DEFAULT key space to 2^{64} at FP 8. Outside that line, ToFA [17] mounts fault attacks from truncated impossible differentials and holds the prior best FP on GIFT-128, reaching FP 5, and the DFA framework MIFA [22] recovers the DEFAULT key at FP 8 from bit-flip faults. A recent *kernel-based* balanced-property miner [13] finds many more balanced properties than the Mixed Integer Linear Programming (MILP) search [25] on which DiFA and DIFA-Rent build, but it cannot search for good input structures and is restricted to small input dimensions.

Most fault attacks, such as DFA and ToFA, use each fault as a single faulty ciphertext, whereas integral fault attacks need a *structure* of $|C|$ ciphertexts, one per distinct fault value, and filter key guesses with *balanced properties*, masks whose parity sum over the partially decrypted structure is zero for the correct key. Assuming a uniform injected value, filling one structure is an instance of the *Coupon-Collector Problem* (CCP) and costs about 53 injections for a nibble fault and 1,567 for a byte fault. We report every fault count under this convention, and N_S structures cost N_S times that figure.

Since the fault propagation is determined entirely by the cipher specification once the fault model is fixed, we assume that a software simulation under the same fault model reproduces the faulty ciphertext of a physical device. Following

recent cryptanalytic work [14,20,22], we validate by simulation the fault count, the computational complexity, and the key-recovery success rate of every attack.

Our Contributions

We propose KIFA (Kernel-based Integral Fault Analysis), an integral fault attack framework that mines its balanced properties with this kernel miner in place of MILP, and compare our contributions with prior work in Table 1.

- **KIFA Framework:** The fault model avoids both limitations of the kernel miner, since it allows only a few small candidate input structures, so only those few need be tried, and each is small enough for the miner to recover its balanced-property space in a single pass. DiFA [18] mines with an MILP search that can decide only single output bits, and DIFA-Rent [12] lifts those bits to higher-weight properties with a one-round linear-structure extension, which is available only when the S-box has a linear structure. The kernel miner finds balanced properties of any weight and needs no S-box-specific structure. On **DEFAULT** and **GIFT-128** the kernel miner’s additional properties improve the attacks. On **GIFT-64** and **BAKSHEESH**, under the byte fault the attacks use, it finds no more properties than DiFA and DIFA-Rent, respectively, but these would need a different method for each cipher, whereas KIFA obtains them with the same miner.
- **Round extension on DEFAULT:** We present a 9-round nibble attack and a 10-round byte attack on **DEFAULT**, at $FP = 9$ and $FP = 10$ under random-position faults. **DEFAULT** is where the kernel miner matters, since the attacks start from 8-round properties under the nibble fault and 9-round properties under the byte fault that it finds, whereas DIFA-Rent, with a nibble fault, finds properties only up to 7 rounds. Gaussian elimination turns these properties into linear combinations that each depend on a single round-key nibble. Under the simple key schedule, in which every **DEFAULT-LAYER** round-key equals the master key, three filtering stages leave a single master-key candidate in all 30 trials from about 1,060 and 12,536 CCP injections, respectively, one and two rounds deeper than the $FP = 8$ of MIFA [22] with two bit-flip faults. Under the rotating key schedule, in which four round-keys derived from the master key alternate, both attacks reduce the key space to 2^{64} , the size DIFA-Rent [12] reaches at $FP = 8$, in all 30 trials from about 848 and 18,804 injections.
- **Round extension on GIFT-128:** We present the first integral fault attacks on **GIFT-128** at $FP = 7$ under a random nibble fault and $FP = 8$ under a random byte fault, improving the prior best $FP = 5$ of ToFA [17]. DiFA [18] reports **GIFT-128** resistant to integral fault attacks because DiFA’s one-round zero-sum filter lets every key pass. The 7-round nibble attack uses the same 5-round properties DiFA finds, one round deeper, and guesses the last two round-keys jointly per QR group. **GIFT-128**’s S-box has no linear structure, so DIFA-Rent’s one-round extension does not apply. The kernel miner additionally finds 38 balanced properties at 6 rounds under a byte

fault, extending the attack to $FP = 8$. DiFA’s bit-level search reaches only 36 of them, and the two further ones make the 8-round byte attack more efficient at the same number of rounds, with fewer trials timing out. The 6-round properties under a byte fault are position-dependent, so the 8-round byte attack fixes the fault at byte 0. The 7-round nibble attack leaves a single master-key candidate in 28 of 30 trials with about 795 CCP injections and in all 30 with 1,590, and the 8-round byte attack in all 30 with 9,402.

- **Round extension on GIFT-64:** We present a 7-round byte attack on GIFT-64 that extends the prior best $FP = 6$ of DiFA [18], counted from its first set of faults, to $FP = 7$ under random and random-position byte faults. The gain comes from the byte fault rather than the miner, since the DiFA framework can also find the balanced properties the attack uses. With $N_S = 64$ byte structures, about 100,288 CCP injections, both variants leave 2^{32} master-key candidates in all 30 trials, which an exhaustive search with one known plaintext-ciphertext pair resolves, and with a second set of byte faults injected two rounds earlier, about 125,360 injections leave a single master-key candidate in the random byte attack.
- **Round extension on BAKSHEESH:** We present an 8-round byte attack on BAKSHEESH that extends DiFA-Rent’s FP from 7 to 8 through a byte fault. Under this fault the kernel miner’s properties span the same space as those that DiFA-Rent’s one-round extension gives, so on BAKSHEESH the gain comes from the fault model rather than the miner. With about 6,268 CCP injections a single master-key candidate remains in 28 of 30 trials and with 15,670 in all 30, and the attack succeeds even under a random-position byte fault.

Although KIFA requires more fault injections than the prior best, reaching more rounds matters more once the fault count stays within practical reach, since the FP determines how many rounds a countermeasure must protect. Recent low-cost voltage fault-injection setups complete campaigns of 10^6 to 10^7 injections within days, of which 10^5 to over 10^6 succeed [10]. All KIFA attacks stay near or below the low end of the successful count, between 636 and 125,360 injections, approximately 2^9 to 2^{17} . Our code is available at <https://anonymous.4open.science/r/KIFA>.

2 Preliminaries

2.1 Notation and Definitions

Bit-vector basics. We work over the binary field $\mathbb{F}_2$. For $x, y \in \mathbb{F}_2^n$, $x \oplus y$ denotes bitwise XOR and $x \cdot m = \bigoplus_{i=0}^{n-1} x_i m_i \in \mathbb{F}_2$ denotes the inner product of x with a mask $m \in \mathbb{F}_2^n$. The *weight* of m is the number of nonzero coordinates of m , equivalently the number of *active bits* of m .

State indexing. A 128-bit state is viewed as $n = 32$ consecutive nibbles indexed $0, \dots, n-1$ from LSB to MSB. Within nibble i , bit $j \in \{0, 1, 2, 3\}$ is global bit $4i + j$. For 64-bit states $n = 16$. We index rounds from 0, so an R -round cipher consists of rounds $0, 1, \dots, R-1$. The input of round r is X_r .

Table 1: Fault attacks on DEFAULT, GIFT-128, GIFT-64, and BAKSHEESH. FP is the number of rounds covered after fault injection, #Faults the injection count, and N_{mk} the remaining master-key candidate space.

Cipher	Attack	FP	Fault Model	#Faults	N_{mk}	Reference
DEFAULT rotating	DIFA-Rent [†]	8	random-position nibble	318	2^{64}	[12]
	MIFA	8	known bit-flip	$11 + \alpha$	1	[22]
	KIFA	9	random-position nibble	848	2^{64}	Sec. 4.1
	KIFA	10	random-position byte	18,804	2^{64}	Sec. 4.1
DEFAULT simple	MIFA	8	known bit-flip	2	1	[22]
	KIFA	9	random-position nibble	636	1.17	Sec. 4.1
	KIFA	9	random-position nibble	1,060	1	Sec. 4.1
	KIFA	10	random-position byte	7,835	1.13	Sec. 4.1
	KIFA	10	random-position byte	12,536	1	Sec. 4.1
GIFT-128	DFA [‡]	3	random nibble	32	2^{17}	[26]
	ToFA	5	random-position byte	3	1	[17]
	KIFA	7	random nibble	795	1.10	Sec. 4.2
	KIFA	7	random nibble	1,590	1	Sec. 4.2
	KIFA	8	random byte	9,402	1	Sec. 4.2
GIFT-64	DiFA [‡]	6	random-position nibble	100	1	[18]
	ToFA	4	random-position byte	22	2^{32}	[17]
	KIFA	7	random byte	100,288	2^{32}	Sec. 4.3
	KIFA	7	random-position byte	100,288	2^{32}	Sec. 4.3
	KIFA [‡]	7	random byte	125,360	1	Sec. 4.3
BAKSHEESH	DFA	3	known bit-flip	12	1	[14]
	ToFA	5	random-position byte	4	1	[17]
	DIFA-Rent [†]	7	random-position nibble	318	1	[12]
	KIFA	8	random byte	6,268	1.13	Sec. 4.4
	KIFA	8	random byte	15,670	1	Sec. 4.4
	KIFA	8	random-position byte	6,268	1.13	Sec. 4.4
	KIFA	8	random-position byte	9,402	1	Sec. 4.4

DEFAULT has 80 rounds, GIFT-128 has 40, GIFT-64 has 28, and BAKSHEESH has 35. N_{mk} for the KIFA rows is the mean over 30 trials, and for the other rows it is the figure reported in the cited work. For DEFAULT, α is the number of additional faults MIFA injects into DEFAULT-CORE. The fault counts of DIFA-Rent and KIFA, which collect fault structures, follow the CCP convention, and that of DiFA is its own CCP estimate.

[†] DIFA-Rent [12] reports the count $N_S \times |\mathcal{C}|$ without CCP overhead, and we re-state this count under the CCP convention, using its $N_S = 6$ nibble structures per cipher, as $6 \times 53 = 318$.

[‡] With a second set of faults injected one round earlier for DFA and two rounds earlier for DiFA and KIFA.

Round keys and equivalent keys. The round-key of the r -th round is K_r . Its *equivalent round-key* is $\overline{K}_r = P^{-1}(K_r)$, for the bit permutation P of Section 2.2.

Linear structures of an S-box. The *linear structures* (LS) of $\mathbf{S}$ form the subspace $\text{LS}(\mathbf{S}) = \{a \in \mathbb{F}_2^4 \mid \mathbf{S}(x \oplus a) \oplus \mathbf{S}(x) \text{ is constant in } x\}$ of dimension $d = \log_2 |\text{LS}(\mathbf{S})|$.

Attack accounting. We write N_S for the number of independent fault structures used in the attack, one per random base plaintext, and N_{mk} for the size of the master-key candidate space remaining after the attack. The fault penetration FP is the number of cipher rounds covered after fault injection, following [12,17].

2.2 GIFT-Family Block Ciphers

This paper analyzes DEFAULT, GIFT-128, GIFT-64, and BAKSHEESH. Each round comprises four operations applied in a fixed order, namely **SubCells**, **PermBits**, **AddRoundConstants**, and **AddRoundKey**.

- **SubCells.** Each round applies a 4-bit S-box $\mathbf{S} : \mathbb{F}_2^4 \rightarrow \mathbb{F}_2^4$, with inverse $\mathbf{S}^{-1}$, in parallel to every nibble; the S-boxes of the four ciphers are listed in Appendix A.
- **PermBits.** The bit permutation P is P_{128} of Figure 1 for DEFAULT, GIFT-128, and BAKSHEESH and its 64-bit downscaling P_{64} for GIFT-64, in both of which the 16 output bits of any 4 consecutive S-boxes feed exactly 4 S-boxes in the next round. This Quotient-Remainder (QR) structure gives 8 QR groups for P_{128} . In QR group g of P_{128} , the *Q-group* is the set $\{4g, 4g+1, 4g+2, 4g+3\}$ of 4 consecutive nibbles before the P-layer, sharing the quotient $\lfloor i/4 \rfloor = g$, and the *R-group* is the set $\{g, g+8, g+16, g+24\}$ of the 4 nibbles where their bits land after the P-layer, sharing the remainder $i \bmod 8 = g$. P_{64} gives 4 QR groups, with Q-group $\{4g, 4g+1, 4g+2, 4g+3\}$ and R-group $\{g, g+4, g+8, g+12\}$ for $g \in \{0, 1, 2, 3\}$.
- **AddRoundConstants.** Each round XORs in a 6-bit round constant.
- **AddRoundKey.** Each round XORs a round-key K_r into every state bit for DEFAULT and BAKSHEESH, but only into bits $4i+1$ and $4i+2$ of each nibble i for GIFT-128 and bits $4i$ and $4i+1$ for GIFT-64.

DEFAULT. It [2] iterates 80 rounds on a 128-bit block with a 128-bit key as $E_{\text{DEFAULT-LAYER}} \circ E_{\text{DEFAULT-CORE}} \circ E_{\text{DEFAULT-LAYER}}$ of 28, 24, and 28 rounds, and our attacks target the last **DEFAULT-LAYER**, rounds $R-28$ to $R-1$ with $R=80$. **SubCells** applies a 4-bit S-box of linear-structure dimension $d=2$. Its linear structures are $\text{LS}(\mathbf{S}) = \{0, 6, 9, \mathbf{f}\}$ and $\text{LS}(\mathbf{S}^{-1}) = \{0, 5, \mathbf{a}, \mathbf{f}\}$. The *simple* key schedule of the first version, described in [20], uses the master key in every **DEFAULT-LAYER** round. The *rotating* key schedule of the published design uses four round-keys in turn, the first being the master key and each next one obtained from the previous one by four **DEFAULT-LAYER** rounds without **AddRoundKey**, so any one of them determines the master key.

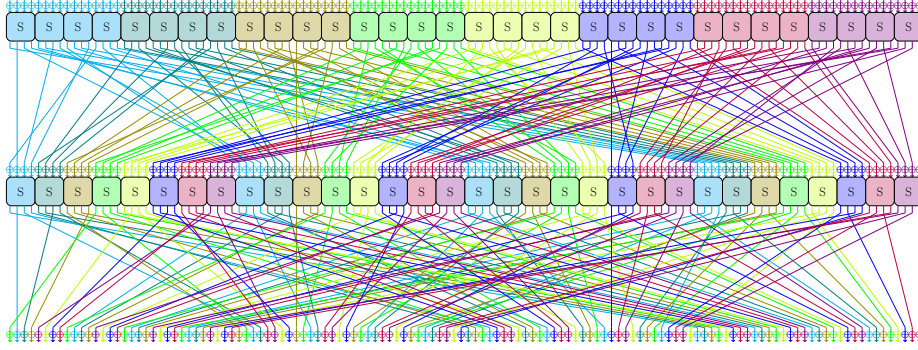


Fig. 1: Bit-permutation P_{128} [5] shared by **DEFAULT**, **GIFT-128**, and **BAKSHEESH**, drawn as in [22]. Two rounds; colors group bits by their original input group.

GIFT-128. It [5] iterates 40 rounds on a 128-bit block with a 128-bit key. **SubCells** applies a 4-bit S-box of linear-structure dimension $d = 0$. The key schedule splits the master key into eight 16-bit words $k_0, \dots, k_7$ and in each round shifts them two positions down, $k'_i = k_{i+2}$ for $i = 0, \dots, 5$, refilling positions 6 and 7 with k_0 and k_1 right-rotated within their 16-bit width by 12 and 2 bits, respectively. Any two consecutive round-keys determine the master key.

GIFT-64. It [5] iterates 28 rounds on a 64-bit block with a 128-bit key. **SubCells** applies the S-box of **GIFT-128**. The key schedule is that of **GIFT-128**, and since each round-key takes 32 bits of the key state, the last four round-keys determine the master key.

BAKSHEESH. It [3] iterates 35 rounds on a 128-bit block with a 128-bit key. **SubCells** applies a 4-bit S-box of linear-structure dimension $d = 1$. Unlike the other three ciphers, **BAKSHEESH** XORs an initial round-key, the master key, into the plaintext, and each next round-key is the previous one rotated right by 1 bit.

2.3 Fault Models Used in This Work

We distinguish two fault models. In a *random* fault the attacker chooses the position and the injected value is random, and in a *random-position* fault the position is random as well, unknown to the attacker but fixed within a fault structure. A random nibble fault replaces one nibble of a state X_r with a value drawn uniformly from $\mathbb{F}_2^4$, and a random byte fault replaces one byte with a value from $\mathbb{F}_2^8$. DiFA [18] and DiFA-Rent [12] use the random-position model and therefore rely only on position-invariant balanced properties, whereas KIFA uses the random model on **GIFT-128**, the random-position model on **DEFAULT**, and both on **GIFT-64** and **BAKSHEESH**. Choosing the position is a modest strengthening, since building a fault structure already requires repeatable injection at one position, and targeting the location is standard for laser and electromagnetic injection.

The byte width is natural even for the nibble-oriented GIFT family, since software stores the 4-bit nibbles in registers of at least 8 bits, and a fault on a register holding two nibbles corrupts both, while a register holding one nibble gives a nibble fault. Bit-sliced implementations match neither model and are outside our scope. Both models are realizable: the random model, at its chosen position, by electromagnetic pulses that confine a fault to one byte on an 8-bit microcontroller [4], and the random-position model by clock glitches on GIFT-128 and GIFT-64 [17].

2.4 Integral Fault Analysis

Replaying the fault at the same plaintext, round, and position collects all its values and forms a fault ciphertext structure $\mathcal{C} = \{C^0, C^1, \dots, C^{|\mathcal{C}|-1}\}$, with $|\mathcal{C}| = 16$ for nibble faults and 256 for byte faults, where C^0 is the fault-free ciphertext. By the coupon-collector argument, filling a structure takes about $|\mathcal{C}| \cdot H_{|\mathcal{C}|-1}$ injections, where $H_n = \sum_{k=1}^n 1/k$, so N_S structures take $N_S \cdot |\mathcal{C}| \cdot H_{|\mathcal{C}|-1}$.

For such a structure, write X_r^i for state X_r during the encryption that produced C^i . A mask $m \in \mathbb{F}_2^{4n}$ is a *balanced property* at X_r if $\bigoplus_{i=0}^{|\mathcal{C}|-1} (X_r^i \cdot m) = 0$ holds for every key and every base plaintext defining $\mathcal{C}$.

We call a fault attack that recovers the key by filtering candidates with balanced (integral) properties [11,16] of a fault structure an *integral fault attack*. DiFA [18] is the first such attack to obtain its balanced properties from the division property [23,24]; DiFA-Rent [12] extends them by one round using the S-box linear structure; and KIFA obtains them from the kernel miner of Section 2.5.

To use a balanced property m at X_r as a key filter, the attacker guesses the round-key bits needed to partially decrypt every C^i back to the active bits of m , computes $\bigoplus_i X_r^i \cdot m$, and keeps the guess if this sum is 0 and discards it otherwise. Repeating this on earlier rounds recovers enough consecutive round-keys to invert the key schedule. On GIFT-128 and GIFT-64 a round-key is XORed into only two bits of each nibble, so we guess the last two round-keys jointly per QR group from $4^4 \times 4^4 = 2^{16}$ candidates, whereas the full-state round-keys of DEFAULT and BAKSHEESH make a joint two-round guess computationally infeasible in practice, so there we guess one round-key at a time.

The standard complexity analysis of integral key-recovery attacks relies on the wrong-key randomization (WKR) hypothesis, under which a wrong key gives a uniformly random parity for each balanced property, so that it survives v independent properties on one structure with probability 2^{-v} and the surviving key space decreases exponentially in N_S . For ciphers including PRESENT, SKINNY, and GIFT, the survival probability of wrong keys deviates from this prediction [8], mainly because of the differential-linear bias [6] of S^{-1} applied during partial decryption. We therefore measure the surviving key space empirically instead of predicting it.

2.5 Kernel-Based Balanced-Property Miner

The kernel-based balanced-property miner of Hwang et al. [13] finds balanced properties directly from random-key parity statistics. Given an active pattern that fixes which plaintext bits vary, the miner samples a fresh random key and a fresh random plaintext multiset for each trial and stacks the bitwise XOR of the ciphertexts within each multiset as a row of the binary parity matrix M . The right kernel $\ker(M)$ contains every key-independent balanced property, and the miner returns a basis of it as candidates for the balanced properties of the input structure, but $\ker(M)$ may also contain spurious masks. Hwang et al. address this with an independent validation step that bounds the false-accept probability of any spurious mask by 2^{-tN} , where t is the number of validation multisets and N is the number of random keys per multiset. KIFA runs this validation with 1,000 random keys and 1,000 structures per key, that is, $tN = 10^6$ key-structure pairs, and every mask it uses passes, so each basis mask is a key filter that the correct key passes.

MILP-based integral distinguisher search [25] decides for each output bit whether it is balanced, so it misses properties that are balanced only as a sum of several bits, whereas the kernel miner returns the whole space of balanced masks within practical input sizes. The kernel miner has two limitations. It cannot scale above a practical input-size range, and it works only on a given input structure rather than searching for one.

3 The KIFA Framework

3.1 Kernel-Based Balanced-Property Miner for Fault Attacks

The two limitations of the kernel miner identified in Section 2.5 disappear in the fault-attack setting. Nibble and byte faults produce structures of size 2^4 and 2^8 , which fall in the practical range of the kernel miner. DIFA-Rent [12] also operates at the same input sizes by combining MILP-derived base properties with a one-round LS extension, but on DEFAULT, on GIFT-128, and under the nibble fault on BAKSHEESH the kernel miner returns more balanced properties at the base-mining step, as we quantify in Section 4. The other limit, that the miner needs a given input structure, is removed by the fault model itself, since the fault position and fault type fix a single input structure.

3.2 Required N_S

When the cipher's S-box has linear-structure dimension $d > 0$, as in DEFAULT ($d = 2$) and BAKSHEESH ($d = 1$), key candidates that differ from the correct key by linear structures of $\mathbb{S}^{-1}$ produce identical filtering responses under the properties of X_{R-1} , so each round-key nibble is determined only up to 2^d values. A set of m round-key nibbles therefore keeps at least 2^{dm} candidates and a round-key of n nibbles at least the lower bound $L^{\min} := 2^{dn}$, so DEFAULT keeps $L^{\min} = 2^{64}$, and BAKSHEESH keeps at least 2^4 candidates per QR group and $L^{\min} = 2^{32}$.

Even when $L^{\min} > 1$, the key schedule together with filtering across multiple attack stages can reduce the surviving master-key space to 1, as we show for **DEFAULT** under its simple key schedule in Section 4.1 and for **BAKSHEESH** in Section 4.4. When the cipher’s S-box is LS-free ($d = 0$), $L^{\min} = 1$.

We determine the required number of fault structures experimentally and report the mean N_{mk} over 30 random trials per choice of N_S . The WKR hypothesis, under which a wrong key fails each mask with probability $1/2$ per structure, may not hold for the **GIFT** family, so we do not use it to predict N_S .

4 Application of KIFA to GIFT-Family Ciphers

This section applies the KIFA framework of Section 3 to four **GIFT**-family ciphers, **DEFAULT**, **GIFT-128**, **GIFT-64**, and **BAKSHEESH**.

4.1 **DEFAULT**: 9-Round Nibble and 10-Round Byte Attacks

In the last **DEFAULT-LAYER**, the 9-round nibble attack injects a random-position nibble fault at X_{R-9} ($\text{FP} = 9$, $|\mathcal{C}| = 2^4$) and the 10-round byte attack a random-position byte fault at X_{R-10} ($\text{FP} = 10$, $|\mathcal{C}| = 2^8$). Under both key schedules the kernel miner finds 96, 64, and 32 balanced properties 6, 7, and 8 rounds after the nibble fault, at X_{R-3} , X_{R-2} , and X_{R-1} . The byte fault gives the same counts one round later, and at X_{R-1} the two faults give the same properties. The kernel miner’s spaces, found for a fault at nibble 0 or byte 0, lie in those of every other position, so their intersection over all positions equals them and the random-position model gives the same properties as a fault fixed at nibble 0 or byte 0, and the attacks need not know the faulted position. Fixing the fault at nibble 21, whose spaces are the largest, does not help, since its extra dimensions are key-independent and leave the result of every trial unchanged. The attacks read the 32 properties at X_{R-1} , the 64 7-round properties under the nibble fault at X_{R-2} , and 64 balanced bits at X_{R-3} , all listed in Appendix B and balanced on 1,000 random keys with 1,000 structures per key under both schedules and at every position.

Stage 1 filters $\overline{K}_{R-1}$ with the 32 properties at X_{R-1} . Every nonzero balanced mask there spans at least four nibbles, and even the 12 weight-4 ones, each on bit 2 of four nibbles such as $\{0, 3, 4, 7\}$, touch two Q-groups and four R-groups, so a property applied directly requires a joint guess of all its nibbles. For a nibble where a property has a 4-bit mask u in the 3-dimensional subspace $\{0, 1, 6, 7, 8, 9, \mathbf{e}, \mathbf{f}\}$ of $\mathbb{F}_2^4$, $u \cdot \mathbf{S}^{-1}$ has algebraic degree at most 1, so $u \cdot \mathbf{S}^{-1}(y \oplus k) = u \cdot \mathbf{S}^{-1}(y) \oplus a \cdot k$ with a depending only on u , and the bit $a \cdot k$, the same for every ciphertext, cancels over a structure of even size. Every other nibble mask of the 32 properties is 2 or 4, and since $2 \oplus 4 = 6$ lies in this subspace, the two give parity sums that differ only by a key-independent term, so each nibble contributes the single key-dependent bit $\bigoplus_y 2 \cdot \mathbf{S}^{-1}(y \oplus k)$. The parity sum of a property therefore depends on the key only through one bit per nibble, and these bits of the 32 properties have rank 32. Gaussian elimination over $\mathbb{F}_2$ thus

Table 2: Empirical N_{mk} for the 9-round nibble and 10-round byte attacks on DEFAULT under random-position faults as a function of N_S (mean over 30 random trials), under the simple and the rotating key schedule.

9-round nibble attack (random-position fault at X_{R-9})				10-round byte attack (random-position fault at X_{R-10})			
N_S	#Faults	Mean N_{mk}	$N_{mk} = 1$ %	N_S	#Faults	Mean N_{mk}	$N_{mk} = 1$ %
<i>Simple key schedule</i>							
8	424	3.30	50.0%	5	7,835	1.13	93.3%
12	636	1.17	83.3%	6	9,402	1.03	96.7%
16	848	1.03	96.7%	8	12,536	1.00	100%
20	1,060	1.00	100%	10	15,670	1.00	100%
24	1,272	1.00	100%	12	18,804	1.00	100%
28	1,484	1.00	100%				
32	1,696	1.00	100%				
40	2,120	1.00	100%				
<i>Rotating key schedule*</i>							
12	636	$2^{64.05}$	96.7%	8	12,536	$2^{64.34}$	86.7%
16	848	2^{64}	100%	12	18,804	2^{64}	100%
20	1,060	2^{64}	100%	16	25,072	2^{64}	100%
24	1,272	2^{64}	100%	20	31,340	2^{64}	100%

#Faults follows the CCP convention. *Under the rotating key schedule the $N_{mk} = 1$ columns give the share of trials with $N_{mk} = 2^{64}$ instead, since no trial goes below 2^{64} .

gives, for every nibble, a linear combination of the properties whose parity sum depends only on that nibble of $\overline{K}_{R-1}$, so Stage 1 guesses each nibble separately among 16 values, which is equivalent to filtering with all 32 properties, and leaves at least $L^{\min} = 2^{64}$ candidates. Under the rotating key schedule the attack stops at this bound, since K_{R-2} , related to K_{R-1} only through a nonlinear map with full diffusion, absorbs every linear-structure shift of K_{R-1} when treated as independent, and using the relation requires enumerating all 2^{64} candidates.

Under the simple key schedule every round uses the same $\overline{K}$, and Stage 2 filters it at X_{R-2} with the 7-round properties, reduced in the same way to one mask per nibble. Each nibble of X_{R-2} depends on the Stage-1 candidates of a single R-group, its own nibble of $\overline{K}$ entering only through S^{-1} and hence only through its linear-structure coset, so Stage 2 filters each R-group's $4^4 = 256$ candidates on its own, leaving at least 16 per R-group, 2^{32} in total. Stage 3 chooses a Stage-2 candidate for one R-group after another, checks each of the 32 balanced bits at X_{R-3} (bold in Table 12) that do not repeat tests of Stage 2, each depending on four or five R-groups, as soon as candidates for all its R-groups are chosen, and moves on as soon as a check fails. The choices for all 8 R-groups that pass every check are the final candidates.

The correct master key survives in every trial of Table 2. Under the simple key schedule the 9-round nibble and 10-round byte attacks leave a single master-key

candidate in all 30 trials from $N_S = 20$ and $N_S = 8$ ($\approx 1,060$ and $12,536$ CCP injections). Under the rotating key schedule they leave exactly 2^{64} master-key candidates in all 30 trials from $N_S = 16$ and $N_S = 12$ (≈ 848 and $18,804$).

The prior best attacks on **DEFAULT** reach $FP = 8$, **DIFA-Rent** [12] with random-position nibble faults and **MIFA** [22] with bit-flip faults. For the nibble fault, **DiFA** [18], which searches for balanced bits, finds 64 balanced bits after 6 rounds and none after 7, and **DIFA-Rent** [12], whose linear-structure extension lifts balanced bits by one round, finds 32 properties after 7 rounds and none after 8. The kernel miner’s spaces after 6, 7, and 8 rounds contain those of **DiFA** and **DIFA-Rent**. Stage 1 of both attacks thus starts from the 8-round properties under the nibble fault and the 9-round ones under the byte fault, which only the kernel miner finds.

4.2 GIFT-128: 7-Round Nibble and 8-Round Byte Attacks

The prior best fault attack on **GIFT-128** is **ToFA** [17] at $FP = 5$. The 7-round nibble attack injects a random nibble fault at nibble 0 of X_{R-7} ($FP = 7$, $|\mathcal{C}| = 2^4$), and the 8-round byte attack a random byte fault at byte 0 of X_{R-8} ($FP = 8$, $|\mathcal{C}| = 2^8$). The kernel miner returns 80 5-round balanced properties at X_{R-2} and 128 4-round ones at X_{R-3} for the 7-round nibble attack, and for the 8-round byte attack, one round deeper, 38 6-round ones at X_{R-2} and 124 5-round ones at X_{R-3} . All of them are weight-1 except two of the 38, which are weight-2, and Appendix C lists the four sets and the number of 6-round properties at every byte position. Unlike on **GIFT-64** and **BAKSHEESH**, where the kernel miner returns the same number of balanced properties at every byte position, this number depends on the fault position, varying from 20 to 38 across the 16 byte positions, so the 8-round byte attack fixes the fault, a realizable fault model, at byte 0, where the number is 38, and has no random-position variant. Under the nibble fault every nibble position gives 80 properties, but nibbles 0 to 15 and 16 to 31 give two different sets, as **DiFA** [18] also observes, which share only 64, so the 7-round nibble attack fixes the fault at nibble 0.

Stage 1 recovers $\bar{K}_{R-1}$ and $\bar{K}_{R-2}$ jointly per QR group. For each QR group $g \in \{0, \dots, 7\}$, the four nibbles of $\bar{K}_{R-1}$ in R-group g together with the four nibbles of $\bar{K}_{R-2}$ in Q-group g determine the four nibbles of X_{R-2} in Q-group g , so the attacker partially decrypts $\mathcal{C}$ through two rounds under each of the $4^4 \times 4^4 = 2^{16}$ joint candidates of the group and discards those with a nonzero parity sum on a mask whose active bits lie in Q-group g . All 80 masks at X_{R-2} of the 7-round nibble attack and 36 of the 38 of the 8-round byte attack are weight-1 and filter a single group in this way. The other two, with active bits $\{24, 57\}$ and $\{25, 120\}$ spanning Q-groups (1, 3) and (1, 7) respectively, filter the joint triples (g_1, g_3, g_7) surviving Stage 1 at QR groups 1, 3, and 7, retaining only triples whose cross-group parity sum is zero. Stage 2 combines the survivors of all 8 QR groups, lifts each to the master key, derives $\bar{K}_{R-3}$ via the key schedule, partially decrypts one further round, and filters with the masks at X_{R-3} .

Table 3 reports the master keys surviving Stage 2; a single one survives in every trial from $N_S = 30$ for the 7-round nibble attack and from $N_S = 6$ for

Table 3: Empirical N_{mk} for the 7-round nibble and 8-round byte attacks on GIFT-128 as a function of N_S (mean over 30 random trials).

N_S	#Faults (CCP)	Mean N_{mk}	$N_{mk} = 1$ %
<i>7-round nibble attack (random nibble fault at nibble 0 of X_{R-7})</i>			
15	795	1.10	93.3%
20	1,060	1.03	96.7%
30	1,590	1.00	100%
50	2,650	1.00	100%
<i>8-round byte attack (random byte fault at byte 0 of X_{R-8})</i>			
6	9,402	1.00	100%
7	10,969	1.00	100%
8	12,536	1.00	100%
10	15,670	1.00	100%
15	23,505	1.00	100%
20	31,340	1.00	100%
30	47,010	1.00	100%

Table 4: The 8-round byte attack on GIFT-128 under each miner’s properties.

N_S	#Faults (CCP)	DiFA (36 properties)		KIFA (38 properties)	
		$N_{mk} = 1$	Timeout	$N_{mk} = 1$	Timeout
4	6,268	11/30	19/30	22/30	8/30
5	7,835	24/30	6/30	29/30	1/30
6	9,402	29/30	1/30	30/30	0/30
7	10,969	30/30	0/30	30/30	0/30

Every column uses the same master keys, the same fault structures and the same fault position, and only the mined property set differs. The $N_{mk} = 1$ columns count the trials leaving a single master-key candidate out of 30, and the Timeout columns count the trials whose Stage-1 candidate space exceeds 2^{22} , the budget of our Stage-2 simulation; every trial either leaves a single candidate or times out.

the 8-round byte attack. KIFA thus presents the first integral fault attacks on GIFT-128, reaching FP = 7 and 8 against the prior best ToFA [17] at FP = 5.

DiFA [18] reports GIFT-128 resistant to integral fault attacks because its one-round zero-sum filter on the balanced properties under a nibble fault lets every key pass. The 7-round nibble attack uses the same properties with the fault injected one round earlier, at X_{R-7} , under a joint guess of $(\bar{K}_{R-1}, \bar{K}_{R-2})$ per QR group. GIFT-128’s S-box is LS-free, so DiFA-Rent’s one-round extension has nothing to extend, and under the byte fault both prior methods return the same 36 properties that DiFA finds, against the 38 the kernel miner returns. The two that DiFA misses are the weight-2 masks, since a search over individual output bits returns only weight-1 masks. Table 4 reports what they are worth.

At $N_S = 4$, DiFA’s 36 properties leave a single master-key candidate in 11 of 30 trials against 22 for KIFA, and the remaining trials time out, 19 against 8.

4.3 GIFT-64: 7-Round Byte Attacks

Under a nibble fault the kernel miner returns only balanced properties that DiFA [18] already reports, so the 7-round byte attack injects a byte fault at X_{R-7} ($FP = 7$, $|\mathcal{C}| = 2^8$), at byte 0 in the random byte attack, every byte position giving the same number of properties, and at a random byte in the random-position byte attack.

The kernel miner finds 30 5-round balanced properties at X_{R-2} , all single bits and listed in Appendix D, and every bit of X_{R-3} and X_{R-4} is balanced. The 5-round properties differ between byte positions and share a 24-dimensional subspace, which the random-position byte attack uses. As on GIFT-128, Stage 1 guesses $(\overline{K}_{R-1}, \overline{K}_{R-2})$ jointly per QR group and filters at X_{R-2} . Stage 2 guesses $(\overline{K}_{R-3}, \overline{K}_{R-4})$ per QR group and filters at X_{R-3} and X_{R-4} , which determines $\overline{K}_{R-3}$, but the properties at X_{R-4} hold for every guess of $\overline{K}_{R-4}$, because every nibble value of X_{R-4} occurs an even number of times in a structure. Both attacks therefore leave the 2^{32} values of $\overline{K}_{R-4}$, and an exhaustive search over them with one known plaintext-ciphertext pair gives the master key. With a second set of byte faults injected two rounds earlier, at byte 0 of X_{R-9} , as in DiFA, the same 5-round properties hold at X_{R-4} and determine $\overline{K}_{R-4}$, which removes the exhaustive search.

Table 5: Empirical N_{mk} for the two 7-round byte attacks on GIFT-64 as a function of N_S (mean over 30 random trials).

N_S	#Faults (CCP)	Mean N_{mk}	$N_{mk} = 2^{32}$ %
<i>Random byte attack (fault at byte 0 of X_{R-7})</i>			
16	25,072	$2^{38.32}$	3.3%
24	37,608	$2^{34.06}$	23.3%
32	50,144	$2^{32.80}$	50.0%
48	75,216	$2^{32.10}$	90.0%
64	100,288	2^{32}	100%
<i>Random-position byte attack (fault at a random byte of X_{R-7})</i>			
16	25,072	$2^{37.84}$	13.3%
24	37,608	$2^{34.53}$	56.7%
32	50,144	$2^{32.96}$	70.0%
48	75,216	$2^{32.05}$	96.7%
64	100,288	2^{32}	100%

The correct master key survives in every trial of Table 5. With $N_S = 64$ structures in the first set and 16 in the second ($\approx 125,360$ CCP injections), the random byte attack leaves a single master-key candidate in all 30 trials. The

prior best attacks on **GIFT-64** are DiFA [18] at $FP = 6$, counted from its first set of random-position nibble faults, which recovers the master key with a second set injected two rounds earlier, and ToFA [17] at $FP = 4$ with random-position byte faults, which leaves 2^{32} candidates. KIFA thus extends the fault penetration on **GIFT-64** from 6 to 7. The gain comes from the byte fault rather than the miner, since the DiFA framework can also find these properties.

4.4 BAKSHEESH: 8-Round Byte Attacks

The 8-round byte attack injects a byte fault at X_{R-8} ($FP = 8$, $|\mathcal{C}| = 2^8$), at byte 0 in the random byte attack, every byte position giving the same number of properties, and at a random byte in the random-position byte attack. The kernel miner returns 32 7-round balanced properties at X_{R-1} and 126 6-round ones at X_{R-2} , listed in Appendix E, and the random byte attack uses all of them. Across the 16 byte positions the 32 7-round properties are identical and the 6-round properties share a 112-dimensional common subspace, and the random-position byte attack uses both.

Stage 1 recovers $\bar{K}_{R-1}$ group by group, since each 7-round property lies in a single R-group, searching the 2^{16} values of the four nibbles of each R-group and keeping those that pass every 7-round property in it. The linear structure leaves at least 2^4 values per R-group, so at least 2^{32} candidates for $\bar{K}_{R-1}$ remain. Stage 2 removes them with the 6-round properties at X_{R-2} , for which the key schedule of **BAKSHEESH** derives $\bar{K}_{R-2}$ from each candidate, so no further key is guessed. Since applying all 126 properties at once would require all 32 nibbles of $\bar{K}_{R-1}$, Stage 2 first combines the Stage-1 survivors separately over the two halves $\{0, 3, 4, 7\}$ and $\{1, 2, 5, 6\}$ of the R-groups, filtering each half with the 28 and 30 properties whose key nibbles it determines, or 28 and 16 in the random-position byte attack, and then applies all 126 properties, or the 112 position-invariant ones, to the merged candidates. This leaves a single $\bar{K}_{R-1}$ and hence, through the key schedule, the master key.

Over 30 random trials per N_S , reported in Table 6, the random byte attack leaves a single master-key candidate in 28 of 30 trials (93.3%) at $N_S = 4$ ($\approx 6,268$ CCP injections) and in all 30 trials at $N_S = 10$, and the random-position byte attack in 26 of 30 trials (86.7%) at $N_S = 4$ and in all 30 trials at $N_S = 6$. At $N_S = 6$ and 8 the random byte attack misses 100% by only one trial, which leaves two candidates; the random-position attack reaches 100% already at $N_S = 6$, although it uses a subspace of the properties, since it runs on other fault bytes and hence other structures.

DIFA-Rent [12] reaches the prior best $FP = 7$ on **BAKSHEESH** with a nibble fault, mining 32 6-round balanced properties. Under the same fault the kernel miner finds 48 6-round properties against DIFA-Rent’s 32, but the 7-round attack leaves exactly the same master-key candidates with either set in every trial, and no 7-round property exists under this fault, so the byte fault extends the fault penetration from 7 to 8. DiFA [18], which searches for balanced individual output bits, finds no 7-round property under the byte fault, since no single bit of X_{R-1} is balanced after 7 rounds, and the 8-round byte attack cannot be built from it.

Table 6: Empirical N_{mk} for the two 8-round byte attacks on BAKSHEESH as a function of N_S (mean over 30 random trials).

N_S	#Faults (CCP)	Mean N_{mk}	$N_{mk} = 1$ %
<i>Random byte attack (fault at byte 0 of X_{R-8})</i>			
4	6,268	1.13	93.3%
6	9,402	1.03	96.7%
8	12,536	1.03	96.7%
10	15,670	1.00	100%
<i>Random-position byte attack (fault at a random byte of X_{R-8})</i>			
4	6,268	1.13	86.7%
6	9,402	1.00	100%

DIFA-Rent’s extension of balanced bits through one round of linear structure, applied to the 6-round bits, recovers the 32 weight-3 7-round properties at X_{R-1} , and applied to the 5-round bits gives a 6-round set at X_{R-2} spanning the same $\mathbb{F}_2$ -space as the kernel miner’s, of rank 126 for the random byte attack and 112 for the random-position byte attack. The two sets therefore impose the same filter and reduce the candidate space to the same size, so on BAKSHEESH the gain comes from the fault model rather than the miner, and KIFA’s advantage here is over DiFA. DIFA-Rent’s extension requires the S-box to have a linear structure, whereas the kernel miner returns the balanced-property space of a structure for any S-box.

5 Conclusion

This paper introduces KIFA, an integral fault attack framework built on the kernel-based balanced-property miner of Hwang et al. [13], whose restriction to small input structures fixed in advance poses no obstacle in the fault-attack setting. On DEFAULT, the kernel miner finds 8-round and 9-round balanced properties that neither DiFA nor DIFA-Rent produces as published, and KIFA attacks the last DEFAULT-LAYER at fault penetrations 9 and 10 with random-position faults, one and two rounds beyond the prior best of 8. These attacks leave a single master-key candidate under the simple key schedule of the first version of DEFAULT and stop at $L^{\min} = 2^{64}$ under the published rotating key schedule, where going below 2^{64} is left open. On GIFT-128, KIFA gives the first integral fault attacks, at fault penetrations 7 and 8 under random nibble and random byte faults, against the prior best of 5 by ToFA [17]. On GIFT-64 and BAKSHEESH a byte fault extends the prior best fault penetration from 6 to 7 and from 7 to 8, leaving 2^{32} master-key candidates on GIFT-64, or a single one with a second set of byte faults in the random byte attack, and both attacks also work under random-position byte faults. All these attacks need at most 125,360 fault injections under the CCP convention, near or below the low end of the 10^5 to over 10^6 successful injections that low-cost setups reach in days [10].

References

1. Agoyan, M., Dutertre, J.M., Mirbaha, A.P., Naccache, D., Ribotta, A.L., Tria, A.: How to flip a bit? In: 2010 IEEE 16th International On-Line Testing Symposium (IOLTS). pp. 235–239. IEEE (2010)
2. Baksi, A., Bhasin, S., Breier, J., Khairallah, M., Peyrin, T., Sarkar, S., Sim, S.M.: DEFAULT: Cipher level resistance against differential fault attack. In: Advances in Cryptology – ASIACRYPT 2021. LNCS, vol. 13091, pp. 124–156. Springer (2021)
3. Baksi, A., Breier, J., Chattopadhyay, A., Gerlich, T., Guilley, S., Gupta, N., Isobe, T., Jati, A., Jedlicka, P., Kim, H., Liu, F., Martinásek, Z., Sakamoto, K., Seo, H., Shiba, R.: BAKSHEESH: Similar yet different from GIFT (and ZORRO). IACR Communications in Cryptology **2**(4) (2026)
4. Balasch, J., Arumí, D., Manich, S.: Design and validation of a platform for electromagnetic fault injection. In: 2017 32nd Conference on Design of Circuits and Integrated Systems (DCIS). pp. 1–6. IEEE (2017)
5. Banik, S., Pandey, S.K., Peyrin, T., Sasaki, Y., Sim, S.M., Todo, Y.: GIFT: A small present: Towards reaching the limit of lightweight encryption. In: Cryptographic Hardware and Embedded Systems – CHES 2017. LNCS, vol. 10529, pp. 321–345. Springer (2017)
6. Bar-On, A., Dunkelman, O., Keller, N., Weizman, A.: DLCT: A new tool for differential-linear cryptanalysis. In: Advances in Cryptology – EUROCRYPT 2019. LNCS, vol. 11476, pp. 313–342. Springer (2019)
7. Barengi, A., Bertoni, G.M., Breveglieri, L., Pellicoli, M., Pelosi, G.: Fault attack on AES with single-bit induced faults. In: 2010 Sixth International Conference on Information Assurance and Security (IAS). pp. 167–172. IEEE (2010)
8. Beierle, C., Leander, G., Pehuda, Y.: When the wrong key lives on: The key-recovery procedure in integral attacks. In: Advances in Cryptology – EUROCRYPT 2026. LNCS, vol. 16546, pp. 33–62. Springer (2026)
9. Biham, E., Shamir, A.: Differential fault analysis of secret key cryptosystems. In: Advances in Cryptology – CRYPTO ’97. LNCS, vol. 1294, pp. 513–525. Springer (1997)
10. Bozzato, C., Focardi, R., Palmarini, F.: Shaping the glitch: Optimizing voltage fault injection attacks. IACR Transactions on Cryptographic Hardware and Embedded Systems **2019**(2), 199–224 (2019)
11. Daemen, J., Knudsen, L., Rijmen, V.: The block cipher Square. In: Fast Software Encryption – FSE ’97. LNCS, vol. 1267, pp. 149–165. Springer (1997)
12. Ghosh, S., Kundu, A.K., Saha, D.: DIFA-Rent: Division property based fault attacks on DEFAULT and BAKSHEESH. Cryptology ePrint Archive, Paper 2025/2053 (2025), <https://eprint.iacr.org/2025/2053>
13. Hwang, Y., Kim, I., Kim, S., Lee, M., Shin, H., Hong, D., Hong, S., Lee, D., Sung, J., Seok, B.: Improving neural-inspired integral distinguishers via a linear-algebraic approach. Cryptology ePrint Archive, Paper 2026/340 (2026), <https://eprint.iacr.org/2026/340>
14. Jana, A., Kundu, A.K., Paul, G.: More vulnerabilities of linear structure Sbox-based ciphers reveal their inability to resist DFA. In: Advances in Cryptology – ASIACRYPT 2024. LNCS, vol. 15491, pp. 168–203. Springer (2024)
15. Kim, Y., Daly, R., Kim, J., Fallin, C., Lee, J.H., Lee, D., Wilkerson, C., Lai, K., Mutlu, O.: Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. In: 2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA). pp. 361–372. IEEE (2014)

16. Knudsen, L., Wagner, D.: Integral cryptanalysis. In: Fast Software Encryption – FSE 2002. LNCS, vol. 2365, pp. 112–127. Springer (2002)
17. Kundu, A.K., Ghosh, S., Aikata, A., Saha, D.: ToFA: Towards fault analysis of GIFT and GIFT-like ciphers leveraging truncated impossible differentials. *IACR Transactions on Cryptographic Hardware and Embedded Systems* **2025**(3), 614–643 (2025)
18. Kundu, A.K., Ghosh, S., Saha, D., Rahman, M.: Divide and rule: DiFA – division property based fault attacks on PRESENT and GIFT. In: Applied Cryptography and Network Security – ACNS 2023. LNCS, vol. 13905, pp. 89–116. Springer (2023)
19. Moro, N., Dehbaoui, A., Heydemann, K., Robisson, B., Encrenaz, E.: Electromagnetic fault injection: Towards a fault model on a 32-bit microcontroller. In: 2013 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC). pp. 77–88. IEEE (2013)
20. Nageler, M., Dobraunig, C., Eichlseder, M.: Information-combining differential fault attacks on DEFAULT. In: Advances in Cryptology – EUROCRYPT 2022. LNCS, vol. 13277, pp. 168–191. Springer (2022)
21. Selmkke, B., Brummer, S., Heyszl, J., Sigl, G.: Precise laser fault injections into 90 nm and 45 nm SRAM-cells. In: Smart Card Research and Advanced Applications – CARDIS 2015. LNCS, vol. 9514, pp. 193–205. Springer (2015)
22. Shin, H., Kim, I., Kim, S., Seok, B., Hong, D., Sung, J., Hong, S., Lee, S., Lee, D.: MIFA: An MILP-based framework for improving differential fault attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems* **2026**(3), 465–488 (2026)
23. Todo, Y.: Structural evaluation by generalized integral property. In: Advances in Cryptology – EUROCRYPT 2015. LNCS, vol. 9056, pp. 287–314. Springer (2015)
24. Todo, Y., Morii, M.: Bit-based division property and application to Simon family. In: Fast Software Encryption – FSE 2016. LNCS, vol. 9783, pp. 357–377. Springer (2016)
25. Xiang, Z., Zhang, W., Bao, Z., Lin, D.: Applying MILP method to searching integral distinguishers based on division property for 6 lightweight block ciphers. In: Advances in Cryptology – ASIACRYPT 2016. LNCS, vol. 10031, pp. 648–678. Springer (2016)
26. Xie, M., Tian, F., Li, J.: Differential fault attack on GIFT. *Chinese Journal of Electronics* **30**(4), 669–675 (2021)

A S-boxes of DEFAULT, GIFT-128, GIFT-64, and BAKSHEESH

Table 7: S-box of DEFAULT-LAYER.

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$S_{\text{DEFAULT}}(x)$	0	3	7	e	d	4	a	9	c	f	1	8	b	2	6	5

Table 8: S-box of GIFT-128 (and GIFT-64).

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$S_{\text{GIFT}}(x)$	1	a	4	c	6	f	3	9	2	d	b	7	5	0	8	e

Table 9: S-box of BAKSHEESH.

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$S_{\text{BAKSHEESH}}(x)$	3	0	6	d	b	5	8	e	c	f	9	2	4	a	7	1

B Balanced-Property Masks of DEFAULT

Table 10: Balanced-property masks at X_{R-1} , 8 rounds after the nibble fault and 9 after the byte fault, for both attacks on **DEFAULT**.

Prop	active bits	Prop	active bits
0	{0, 12, 16, 28, 37, 45, 53, 61, 82, 94, 103, 111, 119, 127}	16	{14, 18, 26, 39, 51, 59, 63, 68, 84, 101, 113, 121, 125}
1	{12, 28, 33, 37, 45, 57, 78, 82, 86, 90, 94, 99, 103, 111, 123}	17	{6, 14, 22, 30}
2	{66, 78, 82, 94}	18	{7, 19, 27, 31, 36, 52, 69, 81, 89, 93, 110, 114, 122}
3	{5, 13, 21, 29, 50, 62, 71, 79, 87, 95, 96, 108, 112, 124}	19	{102, 110, 118, 126}
4	{1, 5, 13, 25, 46, 50, 54, 58, 62, 67, 71, 79, 91, 108, 124}	20	{8, 24, 45, 49, 53, 57, 82, 86, 94, 111, 115, 119, 123}
5	{34, 46, 50, 62}	21	{12, 28, 41, 49, 53, 61, 82, 86, 90, 107, 115, 119, 127}
6	{18, 30, 39, 47, 55, 63, 64, 76, 80, 92, 101, 109, 117, 125}	22	{74, 78, 90, 94}
7	{14, 18, 22, 26, 30, 35, 39, 47, 59, 76, 92, 97, 101, 109, 121}	23	{13, 17, 21, 25, 50, 54, 62, 79, 83, 87, 91, 104, 120}
8	{2, 14, 18, 30}	24	{9, 17, 21, 29, 50, 54, 58, 75, 83, 87, 95, 108, 124}
9	{7, 15, 23, 31, 32, 44, 48, 60, 69, 77, 85, 93, 114, 126}	25	{42, 46, 58, 62}
10	{3, 7, 15, 27, 44, 60, 65, 69, 77, 89, 110, 114, 118, 122, 126}	26	{18, 22, 30, 47, 51, 55, 59, 72, 88, 109, 113, 117, 121}
11	{98, 110, 114, 126}	27	{18, 22, 26, 43, 51, 55, 63, 76, 92, 105, 113, 117, 125}
12	{4, 20, 37, 49, 57, 61, 78, 82, 90, 103, 115, 123, 127}	28	{10, 14, 26, 30}
13	{70, 78, 86, 94}	29	{15, 19, 23, 27, 40, 56, 77, 81, 85, 89, 114, 118, 126}
14	{5, 17, 25, 29, 46, 50, 58, 71, 83, 91, 95, 100, 116}	30	{11, 19, 23, 31, 44, 60, 73, 81, 85, 93, 114, 118, 122}
15	{38, 46, 54, 62}	31	{106, 110, 122, 126}

Table 11: Balanced-property masks at X_{R-2} , 7 rounds after the nibble fault and 8 after the byte fault, for both attacks on DEFAULT under the simple key schedule.

Prop active bits	Prop active bits	Prop active bits	Prop active bits
0 {0, 16, 82}	16 {5, 54, 71}	32 {10, 26}	48 {49, 82, 115}
1 {33, 82, 99}	17 {38, 54}	33 {40, 56, 122}	49 {17, 50, 83}
2 {66, 82}	18 {22, 68, 84}	34 {11, 73, 122}	50 {18, 51, 113}
3 {50, 96, 112}	19 {22, 39, 101}	35 {106, 122}	51 {19, 81, 114}
4 {1, 50, 67}	20 {6, 22}	36 {12, 28, 94}	52 {53, 86, 119}
5 {34, 50}	21 {36, 52, 118}	37 {45, 94, 111}	53 {21, 54, 87}
6 {18, 64, 80}	22 {7, 69, 118}	38 {78, 94}	54 {22, 55, 117}
7 {18, 35, 97}	23 {102, 118}	39 {62, 108, 124}	55 {23, 85, 118}
8 {2, 18}	24 {8, 24, 90}	40 {13, 62, 79}	56 {57, 90, 123}
9 {32, 48, 114}	25 {41, 90, 107}	41 {46, 62}	57 {25, 58, 91}
10 {3, 65, 114}	26 {74, 90}	42 {30, 76, 92}	58 {26, 59, 121}
11 {98, 114}	27 {58, 104, 120}	43 {30, 47, 109}	59 {27, 89, 122}
12 {4, 20, 86}	28 {9, 58, 75}	44 {14, 30}	60 {61, 94, 127}
13 {37, 86, 103}	29 {42, 58}	45 {44, 60, 126}	61 {29, 62, 95}
14 {70, 86}	30 {26, 72, 88}	46 {15, 77, 126}	62 {30, 63, 125}
15 {54, 100, 116}	31 {26, 43, 105}	47 {110, 126}	63 {31, 93, 126}

Table 12: Balanced bits at X_{R-3} , 6 rounds after the nibble fault and 7 after the byte fault, for both attacks on DEFAULT under the simple key schedule, which use the bold ones.

Prop active bits	Prop active bits	Prop active bits	Prop active bits
0 0	16 32	32 64	48 96
1 2	17 34	33 66	49 98
2 4	18 36	34 68	50 100
3 6	19 38	35 70	51 102
4 8	20 40	36 72	52 104
5 10	21 42	37 74	53 106
6 12	22 44	38 76	54 108
7 14	23 46	39 78	55 110
8 16	24 48	40 80	56 112
9 18	25 50	41 82	57 114
10 20	26 52	42 84	58 116
11 22	27 54	43 86	59 118
12 24	28 56	44 88	60 120
13 26	29 58	45 90	61 122
14 28	30 60	46 92	62 124
15 30	31 62	47 94	63 126

C Balanced-Property Masks of GIFT-128

Table 13: 5-round balanced-property masks at X_{R-2} for the 7-round nibble attack on GIFT-128.

Prop	active bits	Prop	active bits	Prop	active bits	Prop	active bits	Prop	active bits	Prop	active bits
0	0	14	5	28	40	42	82	56	84	70	28
1	33	15	38	29	73	43	112	57	117	71	61
2	66	16	68	30	12	44	17	58	52	72	94
3	96	17	101	31	45	45	50	59	85	73	124
4	1	18	36	32	78	46	80	60	24	74	29
5	34	19	69	33	108	47	113	61	57	75	62
6	64	20	8	34	13	48	48	62	90	76	92
7	97	21	41	35	46	49	81	63	120	77	125
8	32	22	74	36	76	50	20	64	25	78	60
9	65	23	104	37	109	51	53	65	58	79	93
10	4	24	9	38	44	52	86	66	88		
11	37	25	42	39	77	53	116	67	121		
12	70	26	72	40	16	54	21	68	56		
13	100	27	105	41	49	55	54	69	89		

Table 14: 4-round balanced-property masks at X_{R-3} for the 7-round nibble attack on GIFT-128.

Prop	active bits	Prop	active bits	Prop	active bits	Prop	active bits	Prop	active bits	Prop	active bits
0	0	22	38	44	40	66	82	88	84	110	122
1	33	23	71	45	73	67	115	89	117	111	27
2	66	24	68	46	106	68	112	90	22	112	28
3	99	25	101	47	11	69	17	91	55	113	61
4	96	26	6	48	12	70	50	92	52	114	94
5	1	27	39	49	45	71	83	93	85	115	127
6	34	28	36	50	78	72	80	94	118	116	124
7	67	29	69	51	111	73	113	95	23	117	29
8	64	30	102	52	108	74	18	96	24	118	62
9	97	31	7	53	13	75	51	97	57	119	95
10	2	32	8	54	46	76	48	98	90	120	92
11	35	33	41	55	79	77	81	99	123	121	125
12	32	34	74	56	76	78	114	100	120	122	30
13	65	35	107	57	109	79	19	101	25	123	63
14	98	36	104	58	14	80	20	102	58	124	60
15	3	37	9	59	47	81	53	103	91	125	93
16	4	38	42	60	44	82	86	104	88	126	126
17	37	39	75	61	77	83	119	105	121	127	31
18	70	40	72	62	110	84	116	106	26		
19	103	41	105	63	15	85	21	107	59		
20	100	42	10	64	16	86	54	108	56		
21	5	43	43	65	49	87	87	109	89		

Table 15: Number of 6-round balanced-property masks at X_{R-2} for the 8-round byte attack on **GIFT-128**, by the byte position of the fault.

Byte position	0	1	2	3	4	5	6	7
# masks	38	38	24	22	34	34	22	20
Byte position	8	9	10	11	12	13	14	15
# masks	38	38	24	22	34	34	22	20

Table 16: 6-round balanced-property masks at X_{R-2} for the 8-round byte attack on **GIFT-128**.

Prop	active bits	Prop	active bits	Prop	active bits	Prop	active bits	Prop	active bits	Prop	active bits
0	64	7	69	14	12	21	17	28	116	35	{25, 120}
1	97	8	8	15	108	22	80	29	21	36	88
2	32	9	41	16	76	23	113	30	84	37	56
3	65	10	104	17	44	24	48	31	117		
4	68	11	9	18	16	25	81	32	52		
5	101	12	72	19	49	26	20	33	85		
6	36	13	40	20	112	27	53	34	{24, 57}		

Table 17: 5-round balanced-property masks at X_{R-3} for the 8-round byte attack on **GIFT-128**.

Prop	active bits	Prop	active bits	Prop	active bits	Prop	active bits	Prop	active bits	Prop	active bits
0	0	21	5	42	10	63	15	84	21	105	59
1	33	22	38	43	43	64	16	85	54	106	56
2	66	23	71	44	40	65	49	86	87	107	89
3	99	24	68	45	73	66	82	87	84	108	27
4	96	25	101	46	106	67	115	88	117	109	28
5	1	26	6	47	11	68	112	89	22	110	61
6	34	27	39	48	12	69	17	90	55	111	94
7	67	28	36	49	45	70	50	91	52	112	127
8	64	29	69	50	78	71	83	92	85	113	124
9	97	30	102	51	111	72	80	93	118	114	29
10	2	31	7	52	108	73	113	94	24	115	62
11	35	32	8	53	13	74	18	95	57	116	95
12	32	33	41	54	46	75	51	96	90	117	92
13	65	34	74	55	79	76	48	97	123	118	125
14	98	35	107	56	76	77	81	98	120	119	30
15	3	36	104	57	109	78	114	99	25	120	63
16	4	37	9	58	14	79	20	100	58	121	60
17	37	38	42	59	47	80	53	101	91	122	93
18	70	39	75	60	44	81	86	102	88	123	31
19	103	40	72	61	77	82	119	103	121		
20	100	41	105	62	110	83	116	104	26		

D Balanced-Property Masks of GIFT-64

Table 18: 5-round balanced-property masks at X_{R-2} for the 7-round byte attacks on **GIFT-64**; every bit of X_{R-3} and X_{R-4} is balanced. The random byte attack uses all of them, and the random-position byte attack uses the bold ones.

Prop	active bits	Prop	active bits	Prop	active bits	Prop	active bits
0	0	8	21	16	25	24	60
1	48	9	52	17	56	25	13
2	1	10	5	18	9	26	44
3	32	11	36	19	40	27	61
4	49	12	53	20	57	28	28
5	16	13	20	21	24	29	45
6	33	14	37	22	41		
7	4	15	8	23	12		

E Balanced-Property Masks of BAKSHEESH

Table 19: 7-round balanced-property masks at X_{R-1} for both 8-round byte attacks on BAKSHEESH.

Prop	active bits	Prop	active bits	Prop	active bits	Prop	active bits
0	{0, 33, 66}	8	{8, 41, 74}	16	{16, 49, 82}	24	{24, 57, 90}
1	{1, 34, 96}	9	{9, 42, 104}	17	{17, 50, 112}	25	{25, 58, 120}
2	{2, 64, 97}	10	{10, 72, 105}	18	{18, 80, 113}	26	{26, 88, 121}
3	{32, 65, 98}	11	{40, 73, 106}	19	{48, 81, 114}	27	{56, 89, 122}
4	{4, 37, 70}	12	{12, 45, 78}	20	{20, 53, 86}	28	{28, 61, 94}
5	{5, 38, 100}	13	{13, 46, 108}	21	{21, 54, 116}	29	{29, 62, 124}
6	{6, 68, 101}	14	{14, 76, 109}	22	{22, 84, 117}	30	{30, 92, 125}
7	{36, 69, 102}	15	{44, 77, 110}	23	{52, 85, 118}	31	{60, 93, 126}

Table 20: 6-round balanced-property masks at X_{R-2} for the 8-round byte attacks on BAKSHEESH. The random byte attack uses all of them, and the random-position byte attack uses the bold ones and the sum of each pair of consecutive plain ones.

Prop	active bits	Prop	active bits	Prop	active bits	Prop	active bits	Prop	active bits
0	0	26	6	52	108	78	20	104	26
1	33	27	39	53	13	79	53	105	59
2	66	28	36	54	46	80	86	106	56
3	99	29	69	55	79	81	119	107	89
4	96	30	102	56	76	82	116	108	122
5	1	31	7	57	{14, 109}	83	21	109	27
6	34	32	8	58	47	84	54	110	28
7	67	33	41	59	44	85	87	111	61
8	64	34	74	60	{77, 110}	86	84	112	94
9	97	35	107	61	15	87	117	113	127
10	2	36	104	62	16	88	22	114	124
11	35	37	9	63	49	89	55	115	29
12	32	38	42	64	82	90	52	116	62
13	65	39	75	65	115	91	85	117	95
14	98	40	72	66	112	92	118	118	92
15	3	41	105	67	17	93	23	119	125
16	4	42	10	68	50	94	24	120	30
17	37	43	43	69	83	95	57	121	63
18	70	44	40	70	80	96	90	122	60
19	103	45	73	71	113	97	123	123	93
20	100	46	106	72	18	98	120	124	126
21	5	47	11	73	51	99	25	125	31
22	38	48	12	74	48	100	58		
23	71	49	45	75	81	101	91		
24	68	50	78	76	114	102	88		
25	101	51	111	77	19	103	121		