SAT and Neural Network Filters for Misinjections in Fault Attacks

Hanbeom Shin[1], Dongwoo Kang[1], Donggeun Kwon[2], Seokhie Hong[3] and HeeSeok Kim[1]

[1] Korea University, Republic of Korea
newonetiger@korea.ac.kr, redr1102@korea.ac.kr, 80khs@korea.ac.kr
[2] Kunsan National University, Republic of Korea
dgkwon@kunsan.ac.kr
[3] SmartM2M, Republic of Korea
shhong@smartm2m.co.kr

Abstract. A misinjection, an injection that misses the fault form an attack assumes, can make an elimination fault attack discard the correct key. They occur when the attacker has not determined the timing of the device, and a random-delay countermeasure wider than the S-box layer can force them even on one who has. This paper presents a SAT filter that removes misinjections without the secret key, one injection at a time. It models the differential propagation from the injected round to the end of the cipher in SAT and rejects an injection when the model has no solution for the assumed fault and the observed ciphertext difference. The paper also presents a neural network filter, which passes an injection when the score of a network trained for the same decision is at least 0.5, and a combined filter, which passes it only when the SAT model of the injection also has a solution. We measure the filters on faults injected by clock glitching into `GIFT-64`, `GIFT-128` and `BAKSHEESH` on a microcontroller, in three settings, a non-profiling attacker without and with a random-delay countermeasure and a profiling attacker with it. The SAT filter rejected none of the 14669 intended faults and, for the non-profiling attacker without the random-delay countermeasure, at least 0.940 of the misinjections at the full-diffusion round, over at least 385 each. For the same attacker it raises the success probability of ToFA on `BAKSHEESH`, which recovers the master key, from 5.2% to 61.2%, and the combined filter raises it to 96.2%, each over 2000 trials.

Keywords: Fault attacks · Differential fault analysis · Misinjection · SAT · Key-independent filtering · Random delay · GIFT · BAKSHEESH

1 Introduction

A fault attack is a physical attack in which faults are injected into a cryptographic device during its execution and the resulting faulty outputs are used to recover secret information [BDL97]. Faults are induced by clock and voltage glitches [BGV11, BFP19], by glitches aimed at a single instruction [YGS+16], by electromagnetic pulses and by laser injection, and only a part of the injection attempts produces a fault the analysis can use. Differential Fault Analysis (DFA), proposed by Biham and Shamir [BS97], injects a fault into one of the last rounds and recovers the secret key by comparing the faulty ciphertext with the correct one. Throughout this paper the target round of a fault is counted from the end of the cipher, so that the target round R means the R-th round from the end.

DFA rests on the premise that the injected fault has the form the analysis assumes, a fault of a given size in a given round. This premise fails in two ways. An attacker

who has not determined the timing of the device has to vary the injection point over a window, and some of the injections then reach another round or corrupt multiple bytes. A random-delay countermeasure [CK09] can make even a fixed injection point reach a different position from one execution to the next, since it inserts a number of dummy operations chosen at random in every execution. We call a fault that misses the assumed form a *misinjection*. Misinjections can be fatal to elimination-based fault attacks such as MIFA [SKK+26] and the truncated impossible differential fault attack ToFA [KGAS25]. These attacks discard, for every fault, the key candidates that contradict the observed difference, so a single misinjection that discards the correct key does not cost one more fault but ends the attack in failure.

An attacker who could tell, before using a fault, whether it is intended or a misinjection would be spared this failure. The decision has to be made without the secret key, since the key is what the attack is trying to recover, and it has to be made for each injection rather than for a batch. Existing techniques do not provide it once the difference has spread over the whole state. The active-cell-count patterns of Piret and Quisquater [PQ03] are derived by hand for the Advanced Encryption Standard (`AES`) and cannot be applied once the difference has spread over the whole state. The methods that identify the position of a fault [BMS12, BSS+20, LJS26] address stream ciphers.

Our Contributions

In this paper we show that a fault attacker can filter out misinjections without the key and one injection at a time, by modelling in SAT the differential propagation from the assumed fault to the observed ciphertext difference ΔC and rejecting the injection when the model has no solution. Deciding feasibility with a solver in this way belongs to the same line of work as the automatic search for impossible differentials [ST17, HLJ+20, HSE23]. MIFA [SKK+26] uses a model of the same kind to keep only the faults whose model has a single solution and takes for granted that every fault has the assumed form, whereas we do not use the faulty ciphertext of an injection whose model has no solution. Table 1 summarizes the main results, and our contributions are the following.

- **A SAT-based misinjection filter.** We present a filter that decides for each injection into a block cipher whether it matches the assumed fault model. The differential propagation from the injected round to the last round is modelled as a SAT instance using only the Difference Distribution Table (DDT) of the S-box and the bit permutation, since the round keys and the round constants cancel in a difference. The filter never rejects an intended fault, which follows from its construction rather than from measurement, and rejects misinjections at a detection rate that we measure. Over the 14669 intended faults of the nine measured combinations, none was rejected. For the non-profiling attacker without the random-delay countermeasure it still rejects at least 0.940 of the misinjections at the full-diffusion round on all three ciphers, where a check of the active cells distinguishes nothing, each rate resting on at least 385 misinjections, and on `BAKSHEESH` it holds one round longer than on `GIFT-128`, whose filter model differs only in the S-box.

- **A neural network filter and a combined filter.** We also propose a neural network filter for the same decision, a classifier trained on simulated differences that gives each injection a score, and a combined filter in which this score follows the SAT filter, so that an injection passes only when the SAT model of the injection has a solution and the score is at least the threshold. Used alone at the threshold of 0.5, the neural network filter discards intended faults in all nine combinations, whereas the SAT filter discards none. On `BAKSHEESH`, where the SAT filter passes misinjections that make the attack discard the correct key, the combined filter at

Table 1: Main results on the nine combinations of cipher, attacker and implementation. Misinj. is the number of misinjections among the faulty ciphertexts measured in Section 3.2, with their share. Detection is the share of them that the SAT filter rejects at the injection round of the attack, the third round from the end on `GIFT-64`, the fourth on `GIFT-128` and the fifth on `BAKSHEESH`, where it rejects no intended fault. The last three columns give the success probability over 2000 trials of the reduced ToFA on the two `GIFT` versions and of ToFA on `BAKSHEESH`, without a filter, after the SAT filter and after the SAT filter followed by the classifier score at $t = 0.5$, as in Table 8. On `BAKSHEESH` three trials of the last column for the profiling attacker exceed the memory limit and do not count as successes, so that value is a minimum.

					Success probability		
Cipher	Attacker	Random delay	Misinj.	Detection	No filter	SAT	SAT and score
`GIFT-64`	non-profiling	no	385 (16%)	0.977	0.5%	94.0%	93.7%
`GIFT-64`	profiling	yes	494 (26%)	1.000	0.0%	97.4%	96.6%
`GIFT-64`	non-profiling	yes	331 (13%)	0.834	26.3%	98.0%	98.4%
`GIFT-128`	non-profiling	no	452 (18%)	0.940	2.7%	92.1%	87.7%
`GIFT-128`	profiling	yes	412 (19%)	1.000	0.3%	42.0%	34.9%
`GIFT-128`	non-profiling	yes	490 (37%)	0.967	0.0%	91.6%	89.6%
`BAKSHEESH`	non-profiling	no	420 (20%)	0.814	5.2%	61.2%	96.2%
`BAKSHEESH`	profiling	yes	523 (25%)	0.765	3.4%	52.4%	83.1%
`BAKSHEESH`	non-profiling	yes	340 (24%)	0.824	2.9%	56.9%	93.7%

0.5 rejects 65 of the 78 misinjections that the SAT filter passes for the non-profiling attacker without the random-delay countermeasure and raises the detection rate from 0.814 to 0.969.

- **Key recovery on silicon faults.** On faults injected by clock glitching into a microcontroller, misinjections leave the success probability of ToFA on `BAKSHEESH`, which recovers the master key, at 5.2%, 2.9% and 3.4% for the non-profiling attacker without and with the random-delay countermeasure and for the profiling attacker. The SAT filter raises it to 61.2%, 56.9% and 52.4%, more than ten times its value without a filter in each of the three, and the combined filter at 0.5 raises it to 96.2%, 93.7% and at least 83.1%, each over 2000 trials. The reduced ToFA on the two `GIFT` versions injects one round closer to the end than ToFA on the same cipher and recovers the last round key rather than the master key. There the SAT filter brings the success probability within one point of its value with every misinjection removed in all six combinations, for example from 0.0% to 97.4% on `GIFT-64` for the profiling attacker under the random-delay countermeasure, each over 2000 trials, while the combined filter stays within one point of it on `GIFT-64` and lowers the success probability on `GIFT-128`. The neural network filter alone restores less than the SAT filter on `GIFT-128` and for the profiling attacker on `BAKSHEESH`, and more for the non-profiling attacker on `BAKSHEESH`.

To the best of our knowledge, rejecting misinjections without the key and one injection at a time has not been reported once the difference of a fault has spread over the whole state, where counting active cells distinguishes nothing. We reject most of them at the full-diffusion round on `GIFT`, and on `BAKSHEESH` one round deeper, with a solver-based feasibility test on three bit-permutation ciphers, restore an elimination attack under a random-delay countermeasure, and measure all of this on silicon faults.

The code, the faults measured on hardware and the results behind every table are available at https://anonymous.4open.science/r/sat-nn-misinjection-filters/.

2 Preliminaries

2.1 Notation, Fault Model, and Misinjection

We call the internal value that an encryption updates the state. For a plaintext, C is the ciphertext of a correct encryption and C' the ciphertext of an encryption of the same plaintext into which a fault was injected. Their difference is $\Delta C = C \oplus C'$. What the attacker obtains is the pair (C, C') for the same plaintext, and the intermediate states of the cipher are not observable.

The round of an injection is given by its position from the end of the cipher. The target round R means that the fault is injected into the input of the R-th round from the end. Fault attacks aim at the last rounds, and a position from the end gives ciphers with different numbers of rounds the same reference. Every injection round in this paper is given in this way, whereas the rounds inside the SAT model of Section 2.5 are numbered in the order of encryption.

A fault model fixes three things, the round at which the fault enters, the part of the state it occupies, and whether the attacker chooses its position and value. The fault model of this paper is a *single byte fault*, in which one byte of the state changes to an arbitrary value, and the attacker chooses neither the byte nor the value. The implementations attacked in this paper apply the S-box to one state byte at a time through a 256-entry table, so a clock or voltage glitch [BGV11, BFP19] that corrupts the byte being processed produces this fault.

We call an injection that changes the output but whose result does not match the assumed fault model a *misinjection*. An injection that leaves the output unchanged is instead called an *ineffective injection*, the counterpart of an ineffective fault [Cla07, DEK+18], and the attacker recognizes it at once since $C' = C$. Whether a fault shows in the output and whether it has the assumed form are two different questions, and a misinjection is a matter of the second. An injection after which the device returns no ciphertext is just as visible. Neither is counted, and every misinjection rate in this paper is a share of the injections that changed the output.

The fault model decides what a misinjection is. The experiments on `GIFT` and `BAKSHEESH` in this paper assume a single byte fault at any position, so a single byte fault is intended whichever byte it corrupts. Any other injection that changes the output is a misinjection, whatever it corrupted.

2.2 Differential Fault Analysis

DFA [BS97] injects a fault into one of the last rounds and recovers the round key from C and C'. Let the last round consist of an S-box layer S, a linear layer L and the XOR of a round key K^*, in that order, and let x be the input of the last S-box layer. The ciphertext is $C = K^* \oplus L(S(x))$. The fault changes this input to x', so the faulty ciphertext under the same round key is $C' = K^* \oplus L(S(x'))$, and the input difference the attack targets is $x \oplus x'$. An attacker who guesses the round key as K inverts the last round and obtains a candidate S-box output and input,

$$\hat{y} = L^{-1}(C \oplus K), \qquad \hat{x} = S^{-1}(\hat{y}).$$

The same computation on C' gives the output and input differences of the S-box layer. The output difference

$$\Delta_{\text{out}} = L^{-1}(C \oplus K) \oplus L^{-1}(C' \oplus K) = L^{-1}(\Delta C)$$

does not depend on the guessed key, whereas the input difference

$$\Delta_{\text{in}}(K) = S^{-1}\big(L^{-1}(C \oplus K)\big) \oplus S^{-1}\big(L^{-1}(C' \oplus K)\big)$$

changes with every guess. The correct guess always gives the real input difference $x \oplus x'$, whereas a wrong guess gives it only when its S-box input is another solution of the same pair of differences. This asymmetry is what allows key guesses to be filtered.

The Difference Distribution Table (DDT) of an S-box is defined for an input difference a and an output difference b as

$$\mathrm{DDT}[a][b] = \#\{\, x : S(x) \oplus S(x \oplus a) = b \,\}.$$

The fault model restricts the input difference $x \oplus x'$ to a set of differences, so a guess K survives only if $\Delta_{\mathrm{in}}(K)$ lies in that set. For one difference a of the set, the entry $\mathrm{DDT}[a][\Delta_{\mathrm{out}}]$ is the number of S-box inputs x with $S(x) \oplus S(x \oplus a) = \Delta_{\mathrm{out}}$, and each such x fixes one key through $K = C \oplus L(S(x))$, so the entry is the number of key candidates that a leaves. An S-box with small entries lets one fault discard more keys, and an S-box with many large entries discards fewer. The amount of filtering one fault provides is set by the distribution of the DDT.

With several faults the attack intersects the candidate sets each fault leaves, and the round key is found when one candidate remains. The correct key always survives an intended fault, since the difference obtained by inverting with the correct key is the one that actually occurred, and the assumed fault can produce it. A misinjection can reverse this. Its real difference can lie outside the differences that the assumed fault can produce, and then even the correct key, which gives this real difference, is discarded at that moment. If a single candidate remains it is a wrong one, so one misinjection that discards the correct key does not waste one fault but ends the attack in failure. This is why an elimination attack has to discard misinjections before it uses the faults.

2.3 Target Ciphers

The experiments target `GIFT` in its two versions and `BAKSHEESH`. Both belong to the `GIFT` family. `BAKSHEESH` uses the bit permutation of `GIFT-128` as its linear layer and a different S-box. The two S-boxes differ in the number of nonzero entries of their DDTs, 99 for `GIFT` and 58 for `BAKSHEESH`, and in their linear structures, which we define first.

Definition 1 (Linear structure)**.** For an n-bit S-box $S : \mathbb{F}_2^n \to \mathbb{F}_2^n$, an element $a \in \mathbb{F}_2^n$ is a *linear structure* (LS) of S if there exists $c \in \mathbb{F}_2^n$ such that

$$S(x) \oplus S(x \oplus a) = c \qquad (\forall\, x \in \mathbb{F}_2^n).$$

In terms of the DDT this means $\mathrm{DDT}[a][c] = 2^n$. Since $a = 0$ is always an LS, an LS with $a \neq 0$ is called *non-trivial*.

`GIFT` [BPP+17] is a lightweight block cipher with a Substitution Permutation Network (SPN) structure whose linear layer is a bit permutation. It has two versions, `GIFT-64` with a 64-bit block and 28 rounds and `GIFT-128` with a 128-bit block and 40 rounds. Both use a 128-bit master key and the same S-box, and that S-box has no non-trivial LS. The round function consists of the following four operations.

- `SubCells`. The 4-bit S-box is applied to every nibble of the state.
- `PermBits`. The bits of the state are moved by a fixed permutation.
- `AddRoundKey`. A round key derived from the master key is XORed into the state.
- `AddRoundConstant`. The round constant is XORed into fixed bit positions of the state.

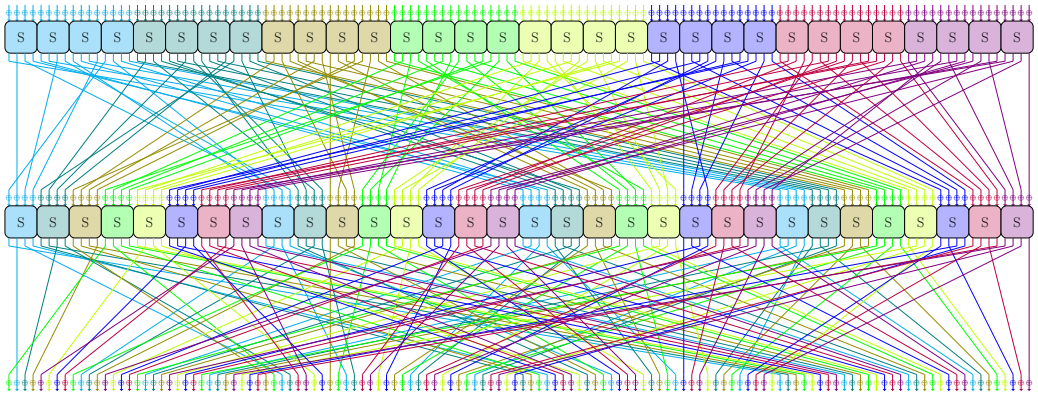


Figure 1: The bit permutation of `GIFT-128` [BPP+17], shared by `BAKSHEESH`, over two rounds. Each colour marks the 16 output bits of one QR group. The XOR before each S-box layer is drawn on every bit, as in `BAKSHEESH`. The figure is redrawn from the diagram in [SKK+26].

`BAKSHEESH` [BBC+23] is a 128-bit block cipher that succeeds `GIFT-128`. It keeps the same bit permutation as its linear layer and reduces the number of rounds from 40 to 35. Apart from the round constants, its round function is that of `GIFT-128` with two differences. `AddRoundKey` XORs the round key into all 128 bits of the state, whereas `GIFT` XORs a round key of half the block size into two fixed bits of every nibble, and the S-box of `SubCells` has one non-trivial LS. The round constants and the key schedules of the two ciphers differ, but their specifications are not needed in what follows and we omit them.

The bit permutations of `GIFT-64` and `GIFT-128`, the latter shared by `BAKSHEESH`, send the four output bits of one S-box to four different S-boxes, so the number of S-boxes that carry a difference grows by up to a factor of four every round. The 16 bits that four neighbouring S-boxes output stay inside four S-boxes of the next round, as Figure 1 shows, and such a group is called a *quotient-remainder* (QR) group [BPP+17], four in `GIFT-64` and eight in `GIFT-128` and `BAKSHEESH`. When two rounds are partially decrypted, the key bits of each QR group can be guessed on their own, and the attacks below reduce the key candidates group by group. The point at which every S-box of the state carries a difference is *full diffusion*. A fault injected at the third round from the end of `GIFT-64`, which has 16 S-boxes, or at the fourth round from the end of `GIFT-128` and `BAKSHEESH`, which have 32, can reach every S-box of the last round, and we call that injection round the *full-diffusion round*.

2.4 ToFA

ToFA [KGAS25] brings impossible differentials into fault attacks on the `GIFT` family. Like DFA it discards the key guesses that contradict the observed difference, but on different grounds. DFA asks whether the input difference obtained under a key guess is one the fault can produce at the last S-box layer, whereas ToFA asks whether the difference the fault produced can reach a given pattern several rounds later. ToFA and a reduced variant of it, described at the end of this section, are the attacks whose success probability we restore in Section 7 by discarding misinjections.

Because of the diffusion above, the difference of a single byte fault can take only a limited set of shapes a few rounds later. Seen as a truncated difference, which records only which bits may carry a difference, the reachable shapes form a finite set that ToFA calls the *permissible truncated difference patterns*. This set is determined by the bit

permutation alone and does not depend on the S-box. The constraint disappears only one round after the difference can first reach every S-box, when every bit of the state may carry a difference. `GIFT-64` therefore has such a set for two rounds after the fault, where every S-box may already carry a difference but only in some of its bits, and `GIFT-128` and `BAKSHEESH` have one for three. Counting the round of the fault as the first, the difference reaches full diffusion at the third round on `GIFT-64` and at the fourth on `GIFT-128` and `BAKSHEESH`, and no pattern constraint remains after those rounds.

The attacker guesses the last two round keys, inverts the ciphertext pair by two rounds and checks whether the resulting difference is in that set. If it is not, the difference cannot have come from the assumed fault and the guess is discarded. The two round keys need not be guessed as a whole. Since the QR groups of Section 2.3 are independent, the attacker guesses the part of the key in each group separately, discards candidates group by group, and then repeats the check on the Cartesian products of the survivors as they are joined. The two inverted rounds plus the rounds of the pattern set give the depth of the fault, so `GIFT-64` with its two-round patterns targets the fourth round from the end and `GIFT-128` and `BAKSHEESH` with their three-round patterns target the fifth.

The reduced candidates for the last two round keys are then taken back through the key schedule to candidates for the round key before them, which the remaining patterns filter again. The original paper recovers the master key of `GIFT-128` uniquely with 3 random byte faults and the master key of `BAKSHEESH` with 4. On `GIFT-64` it reduces the key space to 2^{32} with 22 faults, which is the largest reduction its key schedule allows this way.

The reduced ToFA partially decrypts only the last round and checks the difference against the same permissible patterns, so the fault sits one round closer to the end, at the third round from the end of `GIFT-64` and the fourth of `GIFT-128`. It recovers the last round key, which fixes 32 of the 128 bits of the master key on `GIFT-64` and 64 on `GIFT-128`.

2.5 SAT-Based Search for Differential Trails

Writing a differential trail as constraints and handing it to a solver, instead of following it by hand, is an established method. It was first used to find the best trails. Differential propagation was written as Mixed-Integer Linear Programming (MILP) with the number of active S-boxes as the objective [MWGP11], and the method was extended to bit-oriented ciphers [SHW+14]. Mouha and Preneel [MP13] searched for optimal differential characteristics of ARX ciphers with SAT and satisfiability modulo theories (SMT) solvers. In this use the input and output differences are variables and the trail of highest probability is the objective. Alternatively both can be fixed to specific values and the solver asked only whether any trail connects them. If none exists, the pair of differences cannot occur for any plaintext, and it is an impossible differential. Automatic searches for impossible differentials in this way followed [ST17, HLJ+20, HSE23]. Such models have also been brought to fault attacks. MIFA [SKK+26] writes in MILP the differential propagation from the fault to the ciphertext difference and uses only the faults whose model has a single solution.

The SAT model is the following. Over R consecutive rounds, numbered within the model from 0 in the order of encryption, let X_r be the input difference and Y_r the output difference of the S-box layer of round r, as bit-level binary variables. There are two kinds of constraints. For the S-box layer, the input and output differences of each nibble must be a pair that can occur in the DDT,

$$\mathrm{DDT}\big[X_r^{(j)}\big]\big[Y_r^{(j)}\big] \neq 0 \qquad (j\text{-th nibble}),$$

where $X_r^{(j)}$ and $Y_r^{(j)}$ are the four bits of the j-th nibble, and this constraint is written as the list of allowed pairs. The linear layer is a bit permutation π, so it is the equality

$$Y_r[i] = X_{r+1}[\pi(i)].$$

The round keys and the round constants are XORed and cancel in the difference of two executions, so they do not appear in the model. The size of the model grows with the number of rounds.

2.6 Neural Distinguishers

Gohr [Goh19] showed that a neural network can serve as a differential distinguisher. The network takes a ciphertext pair as input and decides whether the pair was encrypted from a fixed input difference or is a random pair. It is a residual convolutional network. The four 16-bit words of the pair are read as four channels along the bit positions, a convolution of width one combines the words at each bit position, residual blocks of convolutions of width three follow, and two fully connected layers end in a single output between 0 and 1. Its training data are generated by simulation under random keys and plaintexts, so no data from a device are needed to train it. On reduced-round `Speck32/64` the network outperformed the purely differential distinguisher built from the full distribution of output differences. Section 5 uses this kind of network for the decision of this paper, and Section 5.1 states how that task differs from the one above.

3 Misinjections in Practice

In this section we establish the problem before presenting any filter. We first describe the two threat models under which faults are collected and how a random-delay countermeasure affects each of them. We then measure, on hardware, how often an injection misses under each threat model and how much those misinjections reduce the success probability of the attacks on the three ciphers. Finally, we state the problem that a filter has to solve. The SAT filter of Section 4 and the neural network filter of Section 5 solve it in two different ways.

3.1 Threat Models and the Random-Delay Countermeasure

We consider two threat models, the profiling setting [CRR02] and the non-profiling setting, which differ in whether the attacker has a second device of the same kind as the target device. In the profiling setting the attacker has such a device. On that device the attacker determines when the target round executes and which glitch width and offset produce an intended fault, and then applies the same parameters to the target device. The aim is mostly accurate, and we expect few injections to miss. In the non-profiling setting the attacker has only the target device. The timing of the target round is unknown, so the attacker tries injection points over a wide window around the expected one. At some of these points the injection produces a fault other than the assumed one, and such an injection is a misinjection.

A random-delay countermeasure [CK09] can be present under both threat models. It does not prevent an injection but shifts the point in the computation that the injection hits. The random-delay countermeasure inserts dummy operations between the operations of the cipher, and the number of dummy operations is chosen at random in every execution, from 0 up to a maximum that we call the delay bound. A glitch injected at the same moment in every execution can therefore reach a different round or a different cell from one execution to the next. When the device returns a faulty ciphertext, the attacker may no longer know which round received the fault. Misinjections can then arise even for a

profiling attacker who has determined the timing exactly, if the delay is wider than the S-box layer.

The attacker and the implementation vary independently, which gives four combinations. The attacker is either profiling or non-profiling, and the implementation either has the random-delay countermeasure or does not. We measure three of the four on each of the three ciphers, nine combinations in all. On an implementation without the random-delay countermeasure we measure the non-profiling attacker, who varies the injection point over a wide window. On an implementation with the random-delay countermeasure we measure both attackers, the profiling one who injects at the same moment in every execution and the non-profiling one who again varies the injection point. The two combinations with the random-delay countermeasure face different delay bounds, given in Section 3.2, so within one cipher they are two implementations, and their misinjection rates are not compared as if only the attacker changed. We leave out the profiling attacker on an implementation without the random-delay countermeasure, since we expect few of its injections to miss and a filter to have little to do there. The misinjection rate differs between the three combinations, and so does the detection rate of a filter, as Sections 3.2 and 5.3 show. For this reason every measured value in this paper is reported together with the combination it came from.

3.2 Measured Misinjection Rates

Every fault on which a filter or an attack is evaluated in this paper is injected on hardware, and only the training data of the classifier of Section 5 are simulated. The target device is an 8-bit microcontroller, an ATxmega128D4 on a CW308T-XMEGA board, driven by a ChipWhisperer-Husky, and the fault is a clock glitch. The glitch is inserted by XOR into the 7.37 MHz clock of the device, one or two clock cycles long, with a width and an offset within the cycle that are fixed for each cipher and implementation, and its position is an offset in clock cycles from a trigger signal. In the firmware of all three ciphers the S-box layer applies one lookup in a 256-entry table to each byte of the state, so a glitch on one lookup corrupts one byte. The trigger signal is set to cover the S-box layer of the target round, and the glitch is aimed at that interval. On an implementation with the random-delay countermeasure, a loop of n dummy operations is executed between the trigger and the S-box layer of the target round, where n is chosen at random from 0 to the delay bound in every execution. The attacker does not have the secret key and no filter uses it, but to evaluate the filters we decide with the key which injections are misinjections, as follows. For the plaintext of the injection we simulate every possible faulty ciphertext that the assumed single random byte fault at the target round can produce, 8×255 on `GIFT-64` and 16×255 on `GIFT-128` and `BAKSHEESH`, and classify the injection as intended if its faulty ciphertext is in that list and as a misinjection otherwise. An ineffective injection, which leaves the ciphertext unchanged, and an injection that crashes the device give no faulty ciphertext and are not counted, so the counts, misinjection rates and detection rates in this paper are over the injections that produced a faulty ciphertext.

Table 2 lists the faults. Without the random-delay countermeasure 16%, 18% and 20% of the injections of the non-profiling attacker on `GIFT-64`, `GIFT-128` and `BAKSHEESH` were misinjections. A delay narrower than the S-box layer does not stop a profiling attacker. Aimed at the middle of the layer, the injection stays inside it whatever the delay, and under the any-position fault model every single byte fault there is intended. The delay bound of the profiling combination is therefore set wide enough to carry the injection out of the layer, 127 on `GIFT-64` and 255 on `GIFT-128` and `BAKSHEESH`. With these bounds the random-delay countermeasure forces misinjections on the profiling attacker even at the offset from the trigger that gives the most intended faults on the target device itself, and 26%, 19% and 25% of its injections were misinjections. The non-profiling combination with the random-delay countermeasure uses the narrower bounds of 7 on the two `GIFT` versions

Table 2: Misinjection rates of GIFT-64, GIFT-128 and BAKSHEESH in the three combinations of attacker and implementation. The injection round is counted from the end of the cipher. An intended fault is a single random byte fault at any position in the target round, and a misinjection is an injection whose faulty ciphertext no such fault can produce. Injections that left the ciphertext unchanged or crashed the device are not counted, and the rate is the number of misinjections divided by the number of faulty ciphertexts.

Cipher	Round	Attacker	Random delay	Intended	Misinjections	Rate
GIFT-64	3	non-profiling	no	1952	385	16%
GIFT-64	3	profiling	yes	1441	494	26%
GIFT-64	3	non-profiling	yes	2219	331	13%
GIFT-128	4	non-profiling	no	2081	452	18%
GIFT-128	4	profiling	yes	1767	412	19%
GIFT-128	4	non-profiling	yes	831	490	37%
BAKSHEESH	5	non-profiling	no	1679	420	20%
BAKSHEESH	5	profiling	yes	1595	523	25%
BAKSHEESH	5	non-profiling	yes	1104	340	24%

and 15 on BAKSHEESH, and 13%, 37% and 24% of the injections of the non-profiling attacker there were misinjections. A misinjection gives a faulty ciphertext just as an intended fault does, and nothing in the output of the device marks it as a misinjection.

3.3 Effect on Key Recovery

We measure the effect on three key recovery attacks. The attack on BAKSHEESH is the original ToFA [KGAS25], with the fault at the fifth round from the end, and it recovers the master key. The attack on GIFT-64 and GIFT-128 is the reduced ToFA of Section 2.4, with the fault at the third and the fourth round from the end, and it recovers the last round key rather than the master key. The reduced ToFA is a weaker attack than the original. We move the attack closer to the end because the SAT filter works only up to a certain depth, which Section 4.3 measures, and the neural network filter of Section 5 is trained and measured at the same rounds.

For GIFT-64, BAKSHEESH and GIFT-128 we measure the success probability of the attacks in each of the three combinations. In Section 3.2 every fault was injected during the encryption of a random plaintext, so for each injection we have its plaintext, the correct ciphertext and the faulty ciphertext. In one trial we choose m of these injections at random and apply the attack to their correct and faulty ciphertexts. The trial succeeds if exactly one key candidate remains and it is the correct key. We repeat the trial 2000 times for each combination with a new random choice of injections, with $m = 14$ for BAKSHEESH, 30 for GIFT-64 and 22 for GIFT-128. Injecting new faults for every trial would take tens of thousands of injections, whereas the number of ways to choose m of these injections is so large that two trials practically never use the same set of injections. All trials use the same faults, however, so each injection appears in many trials, and the success probability describes these faults. Table 3 gives the success probability of each attack.

When the attack uses all injections, misinjections included, the success probability is at most 26.3%, on GIFT-64 for the non-profiling attacker with the random-delay countermeasure, and on six of the nine combinations it stays below 3%. Without misinjections, the same attacks reach 100.0%, 99.3% and 100.0% on BAKSHEESH, 93.6%, 97.4% and 97.6% on

Table 3: Success probability over 2000 trials of the attacks with all injections, misinjections included, and without misinjections. On `BAKSHEESH` the key search of one trial is limited to 32 GB of memory. A trial that exceeds it might still succeed with more memory but is not counted as a success, so a value with such a trial is a minimum. For the profiling attacker one trial exceeds the limit in the No misinjection column.

Cipher	Attacker	Random delay	m	All	No misinjection
`GIFT-64`	non-profiling	no	30	0.5%	93.6%
`GIFT-64`	profiling	yes	30	0.0%	97.4%
`GIFT-64`	non-profiling	yes	30	26.3%	97.6%
`GIFT-128`	non-profiling	no	22	2.7%	92.8%
`GIFT-128`	profiling	yes	22	0.3%	42.0%
`GIFT-128`	non-profiling	yes	22	0.0%	92.3%
`BAKSHEESH`	non-profiling	no	14	5.2%	100.0%
`BAKSHEESH`	profiling	yes	14	3.4%	99.3%
`BAKSHEESH`	non-profiling	yes	14	2.9%	100.0%

`GIFT-64`, and 92.8%, 42.0% and 92.3% on `GIFT-128`, each triple listing the non-profiling attacker without the random-delay countermeasure, the profiling attacker with it and the non-profiling attacker with it. What the attack loses is therefore not a few points but most of its success probability, and it is the misinjections that take it. The one low value, 42.0% for the profiling attacker on `GIFT-128`, is not a misinjection effect, since it is the value with every misinjection removed.

Adding more faults to one attack does not help. In an elimination-based attack a single misinjection that discards the correct key makes the attack fail, as Section 2.2 explained, so using more faults in one attack raises the chance that one of them is a misinjection that discards the correct key. The misinjections have to be removed before the faults are used in the attack.

3.4 The Filtering Problem

All the attacker obtains from one injection is the ciphertext pair (C, C'). The attacker does not know whether the injected fault has the assumed form and cannot observe the intermediate states, so there is no way to check it directly. The decision has to be made from the ciphertext pair alone.

The decision must not use the secret key, since the key is what the attack is trying to recover. Before full diffusion the decision is easier. A fault of the assumed form can only activate a known set of bits, so checking whether the difference is active only at those bits can already expose a misinjection. After full diffusion every bit can be active, so there is no such pattern to check, and a misinjection is hard to distinguish from an intended fault.

The problem is therefore the following. The input is a ciphertext pair (C, C'), and the output is a decision whether the injection matches the assumed fault model or is a misinjection. The decision can be wrong in two ways, by rejecting an intended fault and by accepting a misinjection.

4 The SAT Filter

This section solves the decision problem of Section 3.4 with a SAT solver. It builds the constraint model, states the two properties that follow from the construction, that

an intended fault is never rejected and that the secret key is not needed, measures on hardware how many misinjections the filter rejects at each injection depth, and shows that on `GIFT-128` and `BAKSHEESH` the S-box sets the depth at which the filter stops working.

4.1 The Constraint Model

The question of Section 3.4 becomes whether a model of the injection has a solution, and we build this model from the SAT model of Section 2.5, whose propagation constraints we keep. The number of rounds R is counted from the round at which the fault is assumed to be injected to the end of the cipher. The output difference is fixed to the observed ΔC, and the input difference is constrained by the fault model. A fault model that is a set of allowed differences in the state at one round can be written as a constraint on the input difference. The rest of this section writes this constraint for the single random byte fault of this paper.

The attacker chooses neither the value nor the position of a single random byte fault. We take the value first. A change of one byte to an arbitrary value means that all 255 nonzero differences in that byte are possible and that every other byte has no difference. This condition takes two constraints. Let $X_0[i]$ be the i-th bit of the input difference of the state and B_j the set of bit positions occupied by the j-th byte. If the j-th byte received the fault, the constraints are

$$X_0[i] = 0 \quad (i \notin B_j), \qquad \sum_{i \in B_j} X_0[i] \geq 1.$$

Together the two constraints make the solutions for X_0 exactly the 255 differences that can occur in that byte, so the assumption of an arbitrary value enters the model without listing the values.

The position is arbitrary as well, and the same model holds it. For each byte we add a binary variable b_j that indicates whether that byte received the fault, and we write the two constraints above per byte. If $b_j = 0$, every bit of the j-th byte is 0, and if $b_j = 1$, at least one bit of it is 1. Adding

$$\sum_j b_j = 1$$

states that exactly one byte is active. Without this sum constraint a difference spread over several bytes would also be a solution, and the injections that the filter is meant to reject would pass. This one line therefore enforces both the arbitrary position and the single byte at once.

We pass this model to CP-SAT, the SAT-based solver of OR-Tools, and ask whether it has a solution. If it has one, the observed ΔC can arise from the assumed fault in the model, and the injection is used in the attack. If it has none, the observed ΔC cannot arise from the assumed fault, and the injection is rejected as a misinjection before the attack sees it. A misinjection passes when the model of its ΔC has a solution, and Section 4.3 measures how often that happens. The filter does not assume anything about what a misinjection looks like, so every misinjection, whatever it corrupted, is checked by the same test. A single byte fault in any byte of the target round passes, since Section 2.1 fixed the position as one the attacker does not choose and any position is intended.

4.2 Properties

The first property is that an intended fault is never rejected. The trail that the fault actually produced is itself a solution of the model, since the fault satisfies the input constraint and the propagation constraints describe every trail the cipher allows. The model therefore has a solution whenever the injection matches the assumed fault. The error of

the filter is one-sided by construction. It can accept a misinjection but it cannot reject an intended fault, and this does not depend on any measurement. The measurements agree. Over the nine combinations of Section 3 the filter examined 14669 intended faults and rejected none of them.

The second property is that the secret key is not needed. Section 2.5 showed that the round keys and the round constants cancel in a difference, so the model contains only the DDT of the S-box and the bit permutation. This meets the requirement of Section 3.4 that the decision does not use the secret key. The solver's answer is also a decision and not a score. The solver returns either a solution or a proof that none exists, so there is no threshold to choose and nothing to adjust to the device.

The cost is one solver call per injection. The model is written once per cipher, with the number of rounds as a parameter, and each injection only changes the fixed output difference. We applied the filter to the faults of the non-profiling combination without the random-delay countermeasure on the three ciphers, one injection after another with nothing else on the machine. The 2337 injections on `GIFT-64` took 75.8 seconds, the 2533 on `GIFT-128` took 229.5 seconds, and the 2099 on `BAKSHEESH` took 210.3 seconds, that is 32.4, 90.6 and 100.2 milliseconds per injection. The machine has an Intel Core i7-12700K processor with 12 cores and 64 GB of memory, and the solver is CP-SAT of OR-Tools 9.15 with 8 workers.

4.3 Detection per Round

We define the detection rate d as the share of the misinjections, labelled as in Section 3.2, that the filter rejects. Since the filter never rejects an intended fault, its pass rate, the share of the intended faults it passes, is 1.000, and the only error is the share $1 - d$ of misinjections that pass.

Table 4 gives d for each injection round, measured for the non-profiling attacker without the random-delay countermeasure, so that the injection depth is the only thing that changes along a row. The round to look at is the full-diffusion round, from which the difference can reach every S-box of the last round. A check that counts active cells distinguishes nothing at this depth, whereas the filter still rejects 0.977 of the misinjections on `GIFT-64`, 0.940 on `GIFT-128` and 0.995 on `BAKSHEESH`. The filter fails one round deeper on the two `GIFT` versions. On `GIFT-64` the detection rate falls to 0.023 at the fourth round from the end, and on `GIFT-128` to 0.000 at the fifth. `BAKSHEESH` holds one round longer. At the fifth round from the end the filter still rejects 0.814 of the misinjections, and its detection rate falls to 0.000 at the sixth. Across the eleven measurements of the table the filter examined 20881 intended faults and rejected none.

The reason the detection rate falls is the propagation itself. One more round of propagation widens the set of differences that the assumed fault can produce, and once that set is wide enough it contains the differences that misinjections produce as well. At that depth the model has a solution for almost every observed ΔC, and the filter can no longer tell the two apart.

4.4 What Sets the Boundary

On `GIFT-128` and `BAKSHEESH` the boundary is set by the S-box. `GIFT-128` and `BAKSHEESH` share the bit permutation, the block size and the fault model, and their implementations apply the S-box as one table lookup per byte of the state, so the one round by which `BAKSHEESH` holds longer in Table 4 comes from the S-box, the only component of the filter's model in which the two ciphers differ. The difference between the two S-boxes shows in their DDTs. Of the 256 entries, 99 are nonzero for the S-box of `GIFT` and 58 for the S-box of `BAKSHEESH`. We expect that the fewer input and output difference pairs an S-box allows, the more often a propagation reaches a contradiction, and the less often

Table 4: Detection rate of the SAT filter for a single random byte fault at any position, per injection round counted from the end. From left to right the values rest on 387, 385, 397 and 378 misinjections on `GIFT-64`, 398, 452 and 388 on `GIFT-128`, and 422, 427, 420 and 423 on `BAKSHEESH`.

	Injection round from the end				
Cipher	2	3	4	5	6
`GIFT-64`	0.964	0.977	0.023	0.000	–
`GIFT-128`	–	0.980	0.940	0.000	–
`BAKSHEESH`	–	0.988	0.995	0.814	0.000

the ΔC of a misinjection can be explained by the assumed fault. The filter on `BAKSHEESH` keeps rejecting misinjections one round deeper than on `GIFT-128`. The two S-boxes also differ in their linear structures, and a comparison of two ciphers does not separate the two properties.

5 The Neural Network Filter

The decision of Section 3.4 can also be learned. A classifier that reads ΔC and outputs a score is the natural alternative to a solver, and this section trains one, explains why it reads ΔC rather than the ciphertext pair, and compares its detection with the solver's on the faults of the nine combinations of Section 3.

5.1 Training a Misinjection Classifier

The task differs from that of the neural distinguisher of Section 2.6 in two ways. There the two classes are ciphertext pairs from a fixed input difference and random pairs, whereas here both classes come from a fault injected into a correct encryption, and the input difference is unknown in both. What separates the classes is only whether the injected fault has the assumed form. A few thousand measured faults are far too few to train on, so the training data are simulated. The trails that the DDT allows from a fault do not depend on the key, but how often each difference occurs does, since the key changes the values that the following S-boxes see. We simulate with every round key set to zero, and Section 5.3 measures whether the classifier transfers to a device with another key.

The training set has 10^6 samples and the validation set 2×10^5, each half intended faults and half misinjections. An intended fault is a single random byte fault at a random position of the target round, as in Section 2.1. A simulated misinjection is one of two kinds in equal numbers, a random fault in two random bytes of the target round, and a single random byte fault injected one round deeper. These two kinds satisfy the definition of a misinjection in Section 2.1, which admits many others, and they are a choice. The misinjections a device produces are not sampled from this distribution, which is one reason the transfer to hardware has to be measured. The input of the classifier is the bit vector of ΔC, 64 bits for `GIFT-64` and 128 bits for the other two ciphers.

We train three architectures. The first is a multilayer perceptron with three hidden layers of 512, 512 and 256 units. The second is a wider and deeper perceptron with hidden layers of 2048, 2048, 1024 and 512 units, trained for twice as many epochs, to check that the first is not too small for the task. The third is a residual convolutional network in the style of Gohr [Goh19], applied along the nibbles of ΔC with the four bits of a nibble as channels. Every network is trained with Adam at a learning rate of 10^{-3}, a cosine schedule and batches of 4096, and the epoch with the lowest validation loss is kept. Each

Table 5: Pass rate of the intended faults and detection rate of the misinjections at the threshold of 0.5 for the two input formats. The rates rest on 1952, 2081 and 1679 intended faults and 385, 452 and 420 misinjections. The difference is read by the residual network on `GIFT-64`, the wider perceptron on `GIFT-128` and the first perceptron on `BAKSHEESH`, and the pair by the residual network on all three. On `GIFT-128` and `BAKSHEESH` the perceptron that reads the difference is a training with another seed than the training of Table 6.

Input	`GIFT-64`	`GIFT-128`	`BAKSHEESH`
ΔC	0.994 / 0.961	0.878 / 0.898	0.661 / 0.852
(C, C')	0.993 / 0.953	0.849 / 0.903	0.641 / 0.802

architecture is trained with three seeds. For every cipher we use one of the nine trainings, the same network for all three combinations, and we choose it without the labels of the measured faults: it is the training with the lowest validation loss among those that give one network for the three combinations. The training data depend only on the cipher, the injection round and the seed, and the two perceptrons train deterministically, so each of their trainings is one network for the three combinations. The residual network does not train deterministically on the GPU, so each combination gets a different network and it cannot be chosen here. It is used in Section 5.2, which compares the inputs on one combination per cipher. The choice is the wider perceptron on `GIFT-64` and `GIFT-128` and the first perceptron on `BAKSHEESH`.

The classifier outputs a score, and a decision needs a threshold. We use the natural threshold of 0.5, which is fixed before any measured fault is seen. No threshold gives the classifier the guarantee that the solver has by construction, to pass every intended fault, and whether a threshold passes every intended fault of a device can only be checked with the labels of the measured faults, which the attacker does not have.

5.2 Input Format

The classifier reads ΔC, as the solver does, so that Section 5.3 compares methods and not inputs. The distinguisher of Gohr [Goh19] reads the ciphertext pair (C, C'), which carries information that the difference does not, so we also train the three architectures on the pair. Since a pair depends on the key, the pair classifier is trained over 2000 random master keys for each of the three kinds of simulated fault, and the ΔC classifier is retrained over the same keys.

Table 5 shows, for the non-profiling attacker without the random-delay countermeasure, that the pair is not a better input than the difference. At the threshold of 0.5 the pair passes fewer intended faults than the difference on all three ciphers, and its detection rate is within 0.008 of the difference's on `GIFT-64` and `GIFT-128` and lower on `BAKSHEESH`, 0.802 against 0.852. Trained over random keys instead of the zero key, the training of Table 6 detects 0.961, 0.887 and 0.836 of the misinjections against 0.958, 0.892 and 0.876, so the key of the simulation changes its detection rate by at most 0.041 here.

5.3 Detection Compared with the Solver

Table 6 puts the two filters side by side on the nine combinations of Section 3. At the threshold of 0.5 the pass rate of the classifier is below 1.000 in all nine, so it discards intended faults that the solver keeps. On the two `GIFT` versions it passes at least 0.903 of them, and on `BAKSHEESH` it passes 0.624 and 0.670 for the non-profiling attacker without and with the random-delay countermeasure and 0.865 for the profiling attacker. Its

Table 6: Pass rate of the intended faults and detection rate of the misinjections for the SAT filter and the classifier at the threshold of 0.5. Intended and Misinj. are the numbers of intended faults and misinjections.

					Pass rate / detection rate	
Cipher	Attacker	Random delay	Intended	Misinj.	Solver	Classifier
GIFT-64	non-profiling	no	1952	385	1.000 / 0.977	0.996 / 0.958
GIFT-64	profiling	yes	1441	494	1.000 / 1.000	0.994 / 1.000
GIFT-64	non-profiling	yes	2219	331	1.000 / 0.834	0.992 / 0.785
GIFT-128	non-profiling	no	2081	452	1.000 / 0.940	0.903 / 0.892
GIFT-128	profiling	yes	1767	412	1.000 / 1.000	0.959 / 0.993
GIFT-128	non-profiling	yes	831	490	1.000 / 0.967	0.911 / 0.953
BAKSHEESH	non-profiling	no	1679	420	1.000 / 0.814	0.624 / 0.876
BAKSHEESH	profiling	yes	1595	523	1.000 / 0.765	0.865 / 0.639
BAKSHEESH	non-profiling	yes	1104	340	1.000 / 0.824	0.670 / 0.874

detection rate is below the solver's in six combinations and equal in one, the profiling attacker on GIFT-64, where both reject every misinjection. It is above the solver's only on BAKSHEESH for the non-profiling attacker, 0.876 against 0.814 without the random-delay countermeasure and 0.874 against 0.824 with it, and there the classifier discards 632 of the 1679 and 364 of the 1104 intended faults.

The difference between the solver and the classifier is not only in the numbers. The solver's pass rate of 1.000 follows from the construction of Section 4.2 and holds on any device and at any depth. The classifier has no such guarantee. Its pass rate depends on the faults the device produces, and whether a threshold passes every intended fault can only be checked with the labels of the measured faults. A threshold that needs no labels of the measured faults can be set just below the lowest score of 10^5 simulated intended faults that the network was not trained on. For the classifier of Table 6 this threshold passes every measured intended fault in all nine combinations. At this threshold the detection rate of the classifier is below the solver's in eight of them and equal only for the profiling attacker on GIFT-64, where both reject all 494 misinjections. On BAKSHEESH the classifier rejects none of the 420, 523 and 340 misinjections of the non-profiling attacker without the random-delay countermeasure, the profiling attacker and the non-profiling attacker with it. The classifier also has to be retrained for every cipher and every depth, whereas the solver takes the number of rounds as a parameter. The two can also be used together, with the score of the classifier applied after the solver, as Section 6 describes.

6 Combining the Two Filters

The classifier of Section 5 gives every injection a score, and the score can follow the SAT filter instead of replacing it. An injection then passes when the SAT filter passes it and its score is at least a threshold t, so the combination still decides for each injection. Since the SAT filter rejects no intended fault, the combination passes the intended faults that the classifier alone passes at t, and every misinjection it rejects beyond the SAT filter is rejected by the score. Table 7 gives the pass rate and the detection rate with the classifier of Table 6 at $t = 0.3$, 0.5 and 0.7.

On the two GIFT versions the score adds little. The SAT filter already rejects at least 0.834 of the misinjections in all six combinations. On BAKSHEESH the score at $t = 0.5$ rejects most of the misinjections that the SAT filter passes, 65 of 78, 65 of 123 and 48 of 60 for the non-profiling attacker without the random-delay countermeasure, the profiling attacker and the non-profiling attacker with it. The detection rate rises from 0.814, 0.765

Table 7: Pass rate of the intended faults and detection rate of the misinjections for the SAT filter followed by the classifier score at t. The SAT filter passes every intended fault, so its column gives the detection rate only. Misinj. is the number of misinjections, and the pass rates rest on 1952, 1441, 2219, 2081, 1767, 831, 1679, 1595 and 1104 intended faults.

				Pass rate / detection rate			
Cipher	Attacker	Random delay	Misinj.	SAT	$t = 0.3$	$t = 0.5$	$t = 0.7$
GIFT-64	non-profiling	no	385	0.977	0.999 / 0.977	0.996 / 0.977	0.979 / 0.979
GIFT-64	profiling	yes	494	1.000	0.999 / 1.000	0.994 / 1.000	0.984 / 1.000
GIFT-64	non-profiling	yes	331	0.834	0.997 / 0.834	0.992 / 0.837	0.977 / 0.852
GIFT-128	non-profiling	no	452	0.940	0.957 / 0.940	0.903 / 0.956	0.797 / 0.971
GIFT-128	profiling	yes	412	1.000	0.981 / 1.000	0.959 / 1.000	0.912 / 1.000
GIFT-128	non-profiling	yes	490	0.967	0.961 / 0.969	0.911 / 0.971	0.816 / 0.976
BAKSHEESH	non-profiling	no	420	0.814	0.873 / 0.917	0.624 / 0.969	0.392 / 0.988
BAKSHEESH	profiling	yes	523	0.765	0.953 / 0.845	0.865 / 0.889	0.772 / 0.920
BAKSHEESH	non-profiling	yes	340	0.824	0.874 / 0.929	0.670 / 0.965	0.419 / 0.991

Table 8: Success probability over 2000 trials. The classifier is the training of Table 6. For the profiling attacker on BAKSHEESH, two trials exceed the memory limit in the Classifier column, three at $t = 0.5$ and 12 at $t = 0.7$, so these values are minimums.

							SAT and score		
Cipher	Attacker	Random delay	m	No filter	SAT	Classifier	$t = 0.3$	$t = 0.5$	$t = 0.7$
GIFT-64	non-profiling	no	30	0.5%	94.0%	94.2%	93.7%	93.7%	94.1%
GIFT-64	profiling	yes	30	0.0%	97.4%	96.6%	97.1%	96.6%	97.9%
GIFT-64	non-profiling	yes	30	26.3%	98.0%	98.0%	97.7%	98.4%	97.6%
GIFT-128	non-profiling	no	22	2.7%	92.1%	89.1%	91.0%	87.7%	85.9%
GIFT-128	profiling	yes	22	0.3%	42.0%	32.7%	37.8%	34.9%	23.8%
GIFT-128	non-profiling	yes	22	0.0%	91.6%	84.7%	89.7%	89.6%	87.0%
BAKSHEESH	non-profiling	no	14	5.2%	61.2%	62.8%	83.6%	96.2%	99.9%
BAKSHEESH	profiling	yes	14	3.4%	52.4%	29.4%	71.4%	83.1%	84.3%
BAKSHEESH	non-profiling	yes	14	2.9%	56.9%	59.6%	83.2%	93.7%	100.0%

and 0.824 to 0.969, 0.889 and 0.965, and the combination pays with intended faults, of which it passes 0.624, 0.865 and 0.670. This cost does not show in the success probability on BAKSHEESH in Section 7, where m is fixed, but the attacker has to inject more often to collect m injections that pass.

7 Key Recovery Results

This section measures what the filters are worth to the attacks of Section 3.3. We compare four configurations, each of which decides for every injection, no filter as in the All column of Table 3, the SAT filter, the classifier at the threshold of 0.5, and the SAT filter followed by the score at $t = 0.3$, 0.5 and 0.7 of Section 6. As in Section 3.3, each value comes from 2000 trials of m injections chosen at random from those the configuration passes, with m fixed for each cipher. Table 8 gives the success probability. On BAKSHEESH the key search of one trial is limited to 32 GB of memory. A trial that exceeds the limit might

still succeed without it but is not counted as a success, so a value with such trials is a minimum.

Without a filter the attacks on the two GIFT versions succeed in at most 26.3% of the trials. With the SAT filter they succeed in 94.0%, 97.4% and 98.0% on GIFT-64 and in 92.1%, 42.0% and 91.6% on GIFT-128, for the non-profiling attacker without the random-delay countermeasure, the profiling attacker and the non-profiling attacker with it, and on all six combinations they come within one point of the value without misinjections in Table 3. The low value for the profiling attacker on GIFT-128 is therefore not a filter effect.

On BAKSHEESH the SAT filter raises the success probability from 5.2%, 3.4% and 2.9% to 61.2%, 52.4% and 56.9% in the same order, against 100.0%, 99.3% and 100.0% without misinjections. We call a misinjection *key-discarding* when, on its own, it makes the check of the attack discard the correct key, which an intended fault never does, and one key-discarding misinjection among the m injections makes the trial fail. Of the 78, 123 and 60 misinjections that the SAT filter passes on BAKSHEESH, 59, 75 and 45 are key-discarding, whereas on the two GIFT versions none of the misinjections it passes is. Counting them needs the correct key, so the count belongs to our evaluation and not to the attacker.

Used in place of the SAT filter, the classifier at the threshold of 0.5 comes within one point of the SAT filter on GIFT-64, is below it on the three GIFT-128 combinations and for the profiling attacker on BAKSHEESH, 29.4% against 52.4%, and is above it for the two non-profiling attackers on BAKSHEESH, 62.8% and 59.6% against 61.2% and 56.9%.

The SAT filter followed by the score gives the largest gain on BAKSHEESH, where the SAT filter leaves key-discarding misinjections. At $t = 0.5$ the success probability rises to 96.2%, 83.1% and 93.7%, and at $t = 0.7$ to 99.9%, 84.3% and 100.0%, where only 3 and 1 of the 2000 trials fail for the two non-profiling attackers. On the two GIFT versions the SAT filter leaves no key-discarding misinjection, so the score can only discard intended faults and misinjections that do not discard the correct key. It stays within one point of the SAT filter on GIFT-64 and lowers the success probability on GIFT-128, most for the profiling attacker, from 42.0% to 23.8% at $t = 0.7$.

8 Conclusion

We showed that a fault attacker can filter out misinjections without the secret key and one injection at a time, by modelling in SAT the differential propagation from the assumed fault to the observed ΔC and rejecting an injection when the model has no solution. The filter never rejects an intended fault, and none of the 14669 measured was rejected. For the non-profiling attacker without the random-delay countermeasure it rejects at least 0.940 of the misinjections at the full-diffusion round, each rate resting on at least 385 misinjections, and on GIFT-128 and BAKSHEESH, whose filter models differ only in the S-box, the S-box decides how deep a fault can still be filtered. On BAKSHEESH, misinjections leave the success probability of ToFA, which recovers the master key, at 5.2%, 3.4% and 2.9% for the non-profiling attacker without the random-delay countermeasure, the profiling attacker and the non-profiling attacker with it. The SAT filter raises it to 61.2%, 52.4% and 56.9%, more than ten times its value without a filter in each, and the score of a neural network classifier after the SAT filter, at the threshold of 0.5, raises it to 96.2%, at least 83.1% and 93.7%, each over 2000 trials. On GIFT the SAT filter brings the reduced ToFA within one point of its success probability with every misinjection removed in all six combinations, a value that for the profiling attacker on GIFT-128 is itself 42.0% of 2000 trials, and the score after it stays within one point of the SAT filter on GIFT-64 and lowers the success probability on GIFT-128. The classifier alone, at the threshold of 0.5, discards intended faults in all nine combinations and restores less than the SAT filter on GIFT-128 and for the profiling attacker on BAKSHEESH and more for the non-profiling attacker on BAKSHEESH.

A random-delay countermeasure can force misinjections on a profiling attacker when its delay is wider than the S-box layer it protects.

Use of AI Tools

We used the generative AI assistant Claude Opus 5.5 (Anthropic) to write code for the experiments and for the analysis behind the tables, to write the paper, and to check spelling and grammar. The authors reviewed every sentence and every number and take full responsibility for the content.

References

[BBC+23] Anubhab Baksi, Jakub Breier, Anupam Chattopadhyay, Tomáš Gerlich, Sylvain Guilley, Naina Gupta, Takanori Isobe, Arpan Jati, Petr Jedlicka, Hyunjun Kim, et al. Baksheesh: similar yet different from gift. *Cryptology ePrint Archive*, 2023.

[BDL97] Dan Boneh, Richard A DeMillo, and Richard J Lipton. On the importance of checking cryptographic protocols for faults. In *International conference on the theory and applications of cryptographic techniques*, pages 37–51. Springer, 1997.

[BFP19] Claudio Bozzato, Riccardo Focardi, and Francesco Palmarini. Shaping the glitch: optimizing voltage fault injection attacks. *IACR transactions on cryptographic hardware and embedded systems*, 2019(2):199–224, 2019.

[BGV11] Josep Balasch, Benedikt Gierlichs, and Ingrid Verbauwhede. An in-depth and black-box characterization of the effects of clock glitches on 8-bit mcus. In *2011 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 105–114. IEEE, 2011.

[BMS12] Subhadeep Banik, Subhamoy Maitra, and Santanu Sarkar. A differential fault attack on the grain family of stream ciphers. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 122–139. Springer, 2012.

[BPP+17] Subhadeep Banik, Sumit Kumar Pandey, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, and Yosuke Todo. Gift: A small present: Towards reaching the limit of lightweight encryption. In *International conference on cryptographic hardware and embedded systems*, pages 321–345. Springer, 2017.

[BS97] Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In *Annual international cryptology conference*, pages 513–525. Springer, 1997.

[BSS+20] Anubhab Baksi, Santanu Sarkar, Akhilesh Siddhanti, Ravi Anand, and Anupam Chattopadhyay. Fault location identification by machine learning. *Cryptology ePrint Archive*, 2020.

[CK09] Jean-Sébastien Coron and Ilya Kizhvatov. An efficient method for random delay generation in embedded software. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 156–170. Springer, 2009.

[Cla07] Christophe Clavier. Secret external encodings do not prevent transient fault analysis. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 181–194. Springer, 2007.

[CRR02] Suresh Chari, Josyula R Rao, and Pankaj Rohatgi. Template attacks. In *International workshop on cryptographic hardware and embedded systems*, pages 13–28. Springer, 2002.

[DEK+18] Christoph Dobraunig, Maria Eichlseder, Thomas Korak, Stefan Mangard, Florian Mendel, and Robert Primas. Sifa: exploiting ineffective fault inductions on symmetric cryptography. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 547–572, 2018.

[Goh19] Aron Gohr. Improving attacks on round-reduced speck32/64 using deep learning. In *Annual International Cryptology Conference*, pages 150–179. Springer, 2019.

[HLJ+20] Xichao Hu, Yongqiang Li, Lin Jiao, Shizhu Tian, and Mingsheng Wang. Mind the propagation of states: New automatic search tool for impossible differentials and impossible polytopic transitions. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 415–445. Springer, 2020.

[HSE23] Hosein Hadipour, Sadegh Sadeghi, and Maria Eichlseder. Finding the impossible: Automated search for full impossible-differential, zero-correlation, and integral attacks. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 128–157. Springer, 2023.

[KGAS25] Anup Kumar Kundu, Shibam Ghosh, Aikata Aikata, and Dhiman Saha. 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.

[LJS26] Kok Ping Lim, Dongyang Jia, and Iftekhar Salam. Deep learning-assisted improved differential fault attacks on lightweight stream ciphers. *arXiv preprint arXiv:2603.29382*, 2026.

[MP13] Nicky Mouha and Bart Preneel. Towards finding optimal differential characteristics for arx: Application to salsa20. *Cryptology ePrint Archive*, 2013.

[MWGP11] Nicky Mouha, Qingju Wang, Dawu Gu, and Bart Preneel. Differential and linear cryptanalysis using mixed-integer linear programming. In *International Conference on Information Security and Cryptology*, pages 57–76. Springer, 2011.

[PQ03] Gilles Piret and Jean-Jacques Quisquater. A differential fault attack technique against spn structures, with application to the aes and khazad. In *International workshop on cryptographic hardware and embedded systems*, pages 77–88. Springer, 2003.

[SHW+14] Siwei Sun, Lei Hu, Meiqin Wang, Peng Wang, Kexin Qiao, Xiaoshuang Ma, Danping Shi, Ling Song, and Kai Fu. Towards finding the best characteristics of some bit-oriented block ciphers and automatic enumeration of (related-key) differential and linear characteristics with predefined properties. *Cryptology ePrint Archive*, 2014.

[SKK+26] Hanbeom Shin, Insung Kim, Sunyeop Kim, Byoungjin Seok, Deukjo Hong, Jaechul Sung, Seokhie Hong, Sangjin Lee, and Dongjae Lee. MIFA: An MILP-based framework for improving differential fault attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2026(3):465–488, 2026.

[ST17] Yu Sasaki and Yosuke Todo. New impossible differential search tool from design and cryptanalysis aspects: Revealing structural properties of several ciphers. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 185–215. Springer, 2017.

[YGS+16] Bilgiday Yuce, Nahid Farhady Ghalaty, Harika Santapuri, Chinmay Deshpande, Conor Patrick, and Patrick Schaumont. Software fault resistance is futile: Effective single-glitch attacks. In *2016 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, pages 47–58. IEEE, 2016.