Modeling Modular Multiplication for Monomial Prediction and Impossible Differentials, with Application to MSX

Emanuele Bellini, Mohamed Rachidi and Sharwan K. Tiwari

Technology Innovation Institute, Abu Dhabi, United Arab Emirates

Abstract

MSX is a recent family of lightweight block ciphers for microcontrollers, and its round function draws nonlinearity from modular addition and modular multiplication on 32-bit words. The multiplication is the distinctive part of the design because it turns the round function into a strongly universal hash function, and from this, the designers prove ideal strong differential and linear properties. The same operation also kept MSX out of reach of automated cryptanalysis, since no exact bit-level model of a modular multiplication was available.

We give two such models for $z = (x \cdot y) \bmod 2^n$. Both work at the bit level and, to the best of our knowledge, both are new. The first is for monomial prediction under the three-subset division property without an unknown subset (3SDP-woU), and it assigns an exact parity to every monomial without using a truth table. The second is a deterministic truncated model for impossible differential search, and we prove that it is not merely sound but exact for two independent operands, so every incompatibility a miss-in-the-middle search reports through a multiplication is genuine.

Both properties are deterministic, so our results are exact, whereas the designers' own analysis provides bounds and estimates, with no explicit trail. Using the two models, we analyze MSX. As far as we know, this is its first independent cryptanalysis. For MSX-128 we obtain a 7-round deterministic integral distinguisher of data complexity 2^{96}, where a 96-bit cube leaves 64 of the 128 output bits balanced, and we extend it to key recoveries on 8 and 9 of the 18 rounds that recover the 256-bit master key in about 2^{228} operations, improving over the MSX designers' generic yoyo key recovery attack on seven rounds. Searching all $16,384$ single-bit pairs at each round count, we also find $1,984$ impossible differentials at seven rounds and none at eight. Both models are implemented in the CLAASP library, so they can be used on any included primitive that uses modular multiplication.

Keywords: Modular multiplication, Monomial prediction, Division property, Impossible differential, Truncated differential, Integral distinguishers, MSX, Automated cryptanalysis, CLAASP

1 Introduction

Most symmetric-key designs draw their nonlinearity from one of two sources, a small S-box or an arithmetic operation on machine words. The arithmetic route is popular because it is fast in software and requires no table; modular addition is by far its most common instance. It is the source of nonlinearity in the addition-rotation-XOR (ARX)

This work is licensed under a "CC BY 4.0" license.
Date of this document: 2026-10-21.

family, in block ciphers such as SPECK [BSS+13], and in stream ciphers and permutations such as ChaCha [Ber08]. Because such designs are so widely used, modular addition is by now well served by automated models. Its monomial propagation under the three-subset division property without unknown subset (3SDP-woU) is captured exactly by the perfect monomial-prediction model of Hu and Yap [HY24], and its deterministic truncated behaviour, which an impossible differential search needs, is captured by the familiar carry propagation rule of an adder. So for an addition based cipher one can point an automated tool at the design and it will work.

Modular multiplication is the rarer choice, and it is not served in the same way. Under the division property the only prior model is that of Xing et al. [XYL18], which tabulates the multiplication as an S-box and hands it to the S-box division-trail algorithm. It works under the weaker two-subset property, and the table confines it to small words, since it was applied only to the 4- and 8-bit multiplications of IDEA-16 and IDEA-32, whereas tabulating a product of two 32-bit variables would need 2^{64} entries. For deterministic truncated differentials, and hence for impossible differential search, we are not aware of any dedicated model at all, and a multiplication is usually either skipped or replaced by a black box. A design that uses a modular multiplication is therefore out of reach of the automated tools, because the component model is missing.

MSX is such a design, and we study it here. Proposed by Minematsu et al. [MSR+26], it is a family of lightweight block ciphers aimed at microcontrollers, with a 64-bit member MSX-64, a two-branch balanced Feistel network, and a 128-bit member MSX-128, a Type-2 four-branch generalized Feistel network, both working on 32-bit branches. It uses no S-box. Its round function applies a 32-bit half multiplication, that is the low half of the product, together with modular additions, and this choice is central to the design. The multiplication makes the round function a strongly universal hash function, and a strongly universal round function forces the maximum expected differential and linear probabilities down to the smallest values they can take. The designers can therefore argue resistance to differential and linear attacks by design rather than by counting trails, an approach they relate to Vaudenay's decorrelation theory [Vau03]. So the whole security argument rests on the multiplication. It is also the operation that no automated model could handle, as we described above, and hence the operation that has kept MSX unanalysed so far.

We therefore model the multiplication, and we do it for two properties which we chose because both are deterministic. An integral distinguisher obtained by monomial prediction holds for every key, and an impossible differential is either present or absent, so in both cases the model returns a fact and not a probability. Differential and linear cryptanalysis are equally fundamental, and the designers treat them first, but there a result takes the form of a bound on the trail probability, whereas a deterministic property can be settled by an explicit construction. For MSX no such construction is available yet, since the designers' analysis reports bounds and estimated round counts and gives no explicit trail. So here we give two exact deterministic models and use them to construct the properties directly.

1.1 Related work

The division property [Tod15] became far more practical once Xiang et al. [XZBL16] expressed its propagation as a mixed-integer linear program, which turned a manual cipher-by-cipher argument into a single solver query. Later work simplified the three-subset rules and added pruning [HW19, WHG+19], removed the unknown subset for public functions to give 3SDP-woU [HLM+20], and showed that a monomial is present exactly when the number of trails between input and output is odd, which makes the analysis exact [HSWW20], with further refinements for large superpolies [HST+21, DL22, HHPW22, HHLW24, HY24]. In all of these tools the nonlinearity of a cipher is reached through S-boxes or through ARX operations, and modular multiplication is not among them.

For the multiplication itself, besides the truth-table model of Xing et al. [XYL18]

discussed above, the only other treatment we are aware of is the bound the MSX designers derive for their own differential and linear trail probabilities [MSR+26]. They write the product bitwise, assume the carry bits have no effect, and bound the chained `AND` operations that remain, and they note themselves that this does not give a tight bound. It serves only those trail probabilities, and their impossible differential estimate uses no model of the multiplication.

Impossible differentials were introduced in [BBS99, Knu98], and they are usually found by the miss-in-the-middle search recalled in Subsection 2.6, which needs a deterministic truncated model of every operation of the cipher. Such models are standard for S-boxes, for XOR, and for modular addition, and our second model provides the same for modular multiplication. A cheaper route is the structural estimate, which counts how far a probability one truncated trail reaches under full diffusion and meets one such trail from each side. MSX follows the Midori design [BBI+15] in its use of this estimate.

MSX was proposed by Minematsu et al. [MSR+26], who also give their own security analysis. They prove that the maximum expected differential and linear probabilities of MSX-64 are $2 \cdot 2^{-64}$ after four rounds, which follows from the round function being strongly universal. For MSX-128, no such bound is known for a non-injective Type-2 Feistel, so they give loose MILP upper bounds instead, and for impossible differentials, they report only an estimated maximum length of eleven rounds. No explicit trail and no concrete distinguisher is given anywhere. Their one key-recovery result is a generic yoyo attack [BLP15] on seven rounds, which recovers the secret round functions rather than the master key. To the best of our knowledge, no independent analysis of MSX has appeared since, so ours is the first.

1.2 Our contributions

We give a bit-level MILP model of $z = (x \cdot y) \bmod 2^n$ for monomial prediction under 3SDP-woU. The product is decomposed into `AND` constraints for the partial products, `COPY` for the repeated operand bits, and the exact modular-addition model for the accumulation, with no truth table. Every monomial receives an exact parity of 0 or 1, so the model supports integral distinguisher detection, ANF and algebraic-degree bounds, and superpoly derivation. To the best of our knowledge, this is the first exact model of modular multiplication under 3SDP-woU.

We give a second model of the same operation, a deterministic truncated model for impossible differential search. The determined region of the output stops at the lowest position where either operand carries an active or unknown difference, and we call this position the pivot. Every output bit below the pivot is a determined zero, every bit from it upward is unknown, and no output bit is ever a determined one. We prove that this rule is not only sound but exact for two independent operands, so every incompatibility a miss-in-the-middle search reports through a multiplication is genuine. We also show where exactness stops, when a single word feeds both operands, a case that does not arise in MSX (Remark 1).

Using the two models, we give the first independent cryptanalysis of MSX. We implement both in the CLAASP library [BGG+23], so they automatically apply to any included primitive that uses modular multiplication. For MSX-128, we find a 7-round deterministic integral distinguisher of data complexity 2^{96}, where a 96-bit cube leaves 64 of the 128 output bits balanced, a consequence of the copy structure of the Type-2 round. We extend the distinguishers to key recoveries on 8 and 9 of the 18 rounds, thereby recovering the 256-bit master key in about 2^{228} time. For impossible differentials we search exhaustively over all $128 \times 128 = 16384$ single-bit pairs at each round count, and find 1984 of them at seven rounds and none at eight, so seven rounds is the maximum the search reaches. Each one is explicit, one active plaintext bit against one active ciphertext bit. The designers estimate the maximum at eleven rounds for MSX-128, from a formula that assumes the

round function diffuses completely and uses no model of the multiplication, and they exhibit no trail. We also derive algebraic-degree bounds. All our results are for round-reduced MSX and do not threaten the full-round security margin.

Table 1 places our results next to the designers' own analysis.

Table 1: Distinguishers and key-recovery attacks on MSX-128-256 (128-bit block, 256-bit key, 18 rounds). Data is the number of chosen plaintexts, and time is in cipher evaluations; a dash marks a quantity that does not apply. The designers' rows are bounds and estimates, and none of them comes with an explicit trail; ours are constructed and deterministic.

Result	Rounds	Data	Time	Source
Differential distinguisher†	8	2^{103}	–	designers [MSR+26]
Impossible differential‡	11	–	–	designers [MSR+26]
(Round) Key recovery	7	2^{98}	2^{136}	designers [MSR+26] +yoyo technique [BLP15]
Integral distinguisher (degree)	6	2^{127}	–	this work
Integral distinguisher (deterministic)	7	2^{96}	–	this work
Integral distinguisher (deterministic)	8	2^{127}	–	this work
Impossible differential (single-bit)	7	–	–	this work
(Master) Key recovery	8	2^{99}	2^{228}	this work
(Master) Key recovery	9	2^{128}	2^{228}	this work

† Probabilistic, and only an upper-bound estimate on the trail probability of 2^{-102}, no explicit trail is given (see Section 7).

‡ An estimate of the maximum length from a diffusion count, with the round function treated as a black box; no trail is given (see Section 6).

1.3 Outline

Section 2 recalls the division property, monomial prediction, the truncated difference encoding, and the propagation rules we build on. Section 3 describes MSX. Section 4 gives the two models of modular multiplication, the monomial-prediction one in Subsection 4.1 and the deterministic truncated one in Subsection 4.2. The three sections that follow report what the models find on MSX: Section 5 the algebraic-degree bounds and the integral distinguishers, Section 6 the impossible differentials, and Section 7 the key-recovery attacks.

2 Preliminaries

2.1 Notations

Let $\mathbb{F}_2 = \{0, 1\}$ and let $\mathbb{F}_2^n$ denote the n-dimensional vector space over $\mathbb{F}_2$. For vectors $\boldsymbol{x} = (x_0, \ldots, x_{n-1})$ and $\boldsymbol{y} = (y_0, \ldots, y_{n-1})$ in $\mathbb{F}_2^n$, we write $\boldsymbol{x} \preceq \boldsymbol{y}$ if $x_i \leq y_i$ for all i, and $\boldsymbol{x} \succeq \boldsymbol{y}$ if $x_i \geq y_i$ for all i. The Hamming weight of $\boldsymbol{x}$, denoted by $\mathrm{wt}(\boldsymbol{x})$, is the number of ones in $\boldsymbol{x}$.

For a Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$, its algebraic normal form (ANF) is defined as

$$f(\boldsymbol{x}) = \bigoplus_{\boldsymbol{u} \in \mathbb{F}_2^n} a_{\boldsymbol{u}}\, \boldsymbol{x}^{\boldsymbol{u}}, \quad a_{\boldsymbol{u}} \in \mathbb{F}_2,$$

where $\boldsymbol{x}^{\boldsymbol{u}} = \prod_{i=0}^{n-1} x_i^{u_i}$ denotes the *monomial* corresponding to $\boldsymbol{u}$. The coefficient $a_{\boldsymbol{u}}$ can be expressed as

$$a_{\boldsymbol{u}} = \bigoplus_{\boldsymbol{x} \preceq \boldsymbol{u}} f(\boldsymbol{x}),$$

and the algebraic degree of f is $\deg(f) = \max_{\boldsymbol{u}: a_{\boldsymbol{u}}=1} \mathrm{wt}(\boldsymbol{u})$.

2.2 Division property

The division property, introduced by Todo [Tod15], traces how monomials propagate through the round functions of a cipher and generalizes integral cryptanalysis. It provides a structured way to reason about the presence or absence of monomials in the ANF of an output bit.

Let $\mathbb{X} \subseteq \mathbb{F}_2^n$ be a multiset of input vectors. For a binary vector $\boldsymbol{u} \in \mathbb{F}_2^n$, the parity of the monomial $\boldsymbol{x}^{\boldsymbol{u}}$ over $\mathbb{X}$ is

$$P_{\boldsymbol{u}} = \bigoplus_{\boldsymbol{x} \in \mathbb{X}} \boldsymbol{x}^{\boldsymbol{u}}.$$

If $P_{\boldsymbol{u}} = 0$ the contribution of $\boldsymbol{x}^{\boldsymbol{u}}$ cancels in the sum, and if $P_{\boldsymbol{u}} = 1$ the monomial is active. For any Boolean function $f(\boldsymbol{x}) = \sum_{\boldsymbol{u}} a_{\boldsymbol{u}}\, \boldsymbol{x}^{\boldsymbol{u}}$, we have

$$\bigoplus_{\boldsymbol{x} \in \mathbb{X}} f(\boldsymbol{x}) \;=\; \sum_{\boldsymbol{u}} a_{\boldsymbol{u}}^{f}\, P_{\boldsymbol{u}},$$

so the effect of each coefficient $a_{\boldsymbol{u}}^{f}$ on the XOR over $\mathbb{X}$ is fully determined by the parities $P_{\boldsymbol{u}}$, and tracking which of them are forced to zero is how one reasons about the monomials that can appear in the ANF.

In the bit-based two-subset division property [TM16], each exponent vector belongs either to a set of unknown parity or to the set whose parity is guaranteed to be zero. Writing $\mathbb{K}$ for the set of exponent vectors of unknown parity, the multiset $\mathbb{X}$ satisfies the two-subset division property $\mathrm{D}_{\mathbb{K}}^{1^n}$ if

$$P_{\boldsymbol{u}} = \bigoplus_{\boldsymbol{x} \in \mathbb{X}} \boldsymbol{x}^{\boldsymbol{u}} = \begin{cases} \text{unknown}, & \text{if } \boldsymbol{u} \succeq \boldsymbol{k} \text{ for some } \boldsymbol{k} \in \mathbb{K}, \\ 0, & \text{otherwise.} \end{cases}$$

One determines $\mathbb{K}$ for the chosen input structure and propagates it through the rounds. If $\mathbb{K}$ propagates to $\mathbb{K}'$ and the unit vector $\mathbf{e}_i$ of an output bit y_i is missing from $\mathbb{K}'$, then $P_{\mathbf{e}_i} = 0$, so $\bigoplus_{\boldsymbol{x} \in \mathbb{X}} y_i = 0$ and y_i is balanced over the structure [TIHM18].

The three-subset variant [TM16, WHG+19] refines this by also tracking monomials of guaranteed odd parity, recovering integral properties that the two-subset form misses. For a public function the unknown subset is unnecessary, since Hao *et al.* [HLM+20] observed that if the round function and the key schedule are treated as a single public map, key bits become input variables, every monomial then has a fixed ANF coefficient, and no parity uncertainty remains. Removing the unknown subset gives the *three-subset division property without unknown subset* (3SDP-woU), which our analysis uses.

Formally, let $\widetilde{\mathbb{L}}$ be a multiset of n-bit exponent vectors. The multiset $\mathbb{X}$ satisfies the 3SDP-woU, denoted $\mathrm{D}_{\widetilde{\mathbb{L}}}^{1^n}$, if for each $\boldsymbol{u} \in \mathbb{F}_2^n$,

$$P_{\boldsymbol{u}} = \bigoplus_{\boldsymbol{x} \in \mathbb{X}} \boldsymbol{x}^{\boldsymbol{u}} = \begin{cases} 1, & \text{if } \boldsymbol{u} \text{ appears an odd number of times in } \widetilde{\mathbb{L}}, \\ 0, & \text{otherwise.} \end{cases}$$

This key-independent formulation lets the propagation of monomial parities be modeled purely algebraically and encoded as MILP constraints. We recall here only what the modular multiplication model and the MSX analysis need; the full framework and its CLAASP implementation are given in [BRT26].

2.3 Monomial prediction

Monomial prediction (MP) [HSWW20] is the algebraic form of 3SDP-woU that we use. For $\boldsymbol{f} : \mathbb{F}_2^n \to \mathbb{F}_2^m$, expand a product of output bits $\boldsymbol{y}^{\boldsymbol{v}} = \prod_i f_i(\boldsymbol{x})^{v_i}$ in the input variables,

$\boldsymbol{y}^{\boldsymbol{v}} = \bigoplus_{\boldsymbol{u}} a_{\boldsymbol{u},\boldsymbol{v}}\, \boldsymbol{x}^{\boldsymbol{u}}$. We write $\boldsymbol{x}^{\boldsymbol{u}} \to \boldsymbol{y}^{\boldsymbol{v}}$ when $a_{\boldsymbol{u},\boldsymbol{v}} = 1$, that is, when the input monomial $\boldsymbol{x}^{\boldsymbol{u}}$ appears in the ANF of $\boldsymbol{y}^{\boldsymbol{v}}$.

When $\boldsymbol{f} = \boldsymbol{f}^{(r-1)} \circ \cdots \circ \boldsymbol{f}^{(0)}$ is written as a composition of round operations, an *r-round monomial trail* is a sequence $(\boldsymbol{x}^{(0)})^{\boldsymbol{u}^{(0)}} \to \cdots \to (\boldsymbol{x}^{(r)})^{\boldsymbol{u}^{(r)}}$ in which every step is a valid monomial propagation through the corresponding operation. A single trail only witnesses that the input monomial *may* reach the output; since several trails can end at the same monomial, their number modulo two decides presence. Writing $\mathsf{Hull}(\boldsymbol{u}^{(0)}, \boldsymbol{u}^{(r)})$ for the set of all such trails, the monomial prediction criterion [HSWW20] is

$$a_{\boldsymbol{u}^{(0)},\boldsymbol{u}^{(r)}} = \left|\mathsf{Hull}(\boldsymbol{u}^{(0)}, \boldsymbol{u}^{(r)})\right| \bmod 2,$$

so $\boldsymbol{x}^{\boldsymbol{u}^{(0)}}$ appears in the ANF of $\boldsymbol{x}^{\boldsymbol{u}^{(r)}}$ exactly when the number of trails is odd. Under 3SDP-woU this parity is expressed by linear constraints and counted by the MILP model. Preserving it exactly is the requirement our modular multiplication model must meet (Subsection 4.1).

2.4 Algebraic attacks based on the division property and monomial prediction

Let an output bit of a keyed primitive be $f : \mathbb{F}_2^n \times \mathbb{F}_2^m \to \mathbb{F}_2$, $(\boldsymbol{x}, \boldsymbol{k}) \mapsto f(\boldsymbol{x}, \boldsymbol{k})$, with $\boldsymbol{x}$ the public input (plaintext or IV) and $\boldsymbol{k}$ the key. Fix a set I of public coordinates, write $\overline{I}$ for its complement, and let $\mathcal{C}_I$ be the cube obtained by letting $\boldsymbol{x}_I$ range over all values while $\boldsymbol{x}_{\overline{I}}$ is fixed to a constant $\boldsymbol{c}$. Writing $f = S_{\mathcal{C}_I}(\boldsymbol{k})\, \boldsymbol{x}_I + Q$ with $\boldsymbol{x}_I = \prod_{i \in I} x_i$ and Q not divisible by $\boldsymbol{x}_I$, summing over the cube isolates the *superpoly*

$$\bigoplus_{\boldsymbol{x}_I \in \mathcal{C}_I} f(\boldsymbol{x}_I, \boldsymbol{c}, \boldsymbol{k}) = S_{\mathcal{C}_I}(\boldsymbol{k}).$$

If $S_{\mathcal{C}_I} \equiv 0$ or $S_{\mathcal{C}_I} \equiv 1$, the cube sum is constant in the key and f is an integral distinguisher; otherwise $S_{\mathcal{C}_I}$ is a key-dependent polynomial whose evaluations give relations on $\boldsymbol{k}$, and when it is simple these relations recover key bits [DS09]. A sufficient condition for the constant case is degree, since if $\deg_{\boldsymbol{x}} f \le d$ then f sums to zero over every affine subspace of dimension $d+1$ [DKR97, Knu94], and division-property degree bounds are typically tighter than classical ones [BC13] because they track individual monomials rather than only the global degree. When several output bits are balanced over a low-dimensional cube, one appends rounds and guesses the added round keys, keeping only guesses consistent with the balance, to turn the distinguisher into a key recovery. Our analysis works in the constant case. Recovering a non-constant superpoly needs the same model but a different search, and we do not pursue it here.

2.5 Propagation rules

The monomial prediction framework requires a precise description of how basic Boolean operations affect the propagation of monomials between their input and output variables. Under the 3SDP-woU of Subsection 2.2, the propagation rules for the basic Boolean operations [XZBL16, HLM+20] describe how the multiset $\widetilde{\mathbb{L}}$ of exponent vectors evolves through each operation, and they form the basis of the MILP model used in CLAASP. Each rule specifies which pairs of input and output exponent vectors are valid, allowing monomial trails to be encoded by linear constraints.

For each operation, a pair $(\boldsymbol{u}, \boldsymbol{v})$ is *valid* if the monomial $\boldsymbol{x}^{\boldsymbol{u}}$ contributes to $\boldsymbol{y}^{\boldsymbol{v}}$ in the algebraic expansion; the set of all such pairs constitutes its propagation rule. Because any Boolean function can be decomposed into `XOR`, `AND`, `NOT`, and `COPY`, the propagation rules of these basic operations determine all monomial trails. We use binary MILP variables $a_1, a_2, \ldots$ for the input exponents and $b, b_1, b_2, \ldots$ for the output exponents.

The valid exponent transitions of the four operations are then:

- `XOR`, $y = x_1 \oplus x_2$: $(0,0) \to 0$, $(1,0) \to 1$, $(0,1) \to 1$, encoded by $b = a_1 + a_2$.
- `AND`, $y = x_1 x_2$: $(0,0) \to 0$, $(1,1) \to 1$, encoded by $b = a_1 = a_2$.
- `NOT`, $y = x \oplus 1$: $0 \to 0$, $0 \to 1$, $1 \to 1$, encoded by $b \geq a$.
- `COPY`, $(y_1, y_2) = (x, x)$: $0 \to (0,0)$ and $1 \to (1,0), (0,1), (1,1)$, encoded by $a \geq b_1$, $a \geq b_2$, $b_1 + b_2 \geq a$.

The modular multiplication model of Subsection 4.1 is built entirely from these rules, namely `AND` for the partial products and `COPY` for the repeated use of the operand bits, together with the exact modular-addition model.

2.6 Impossible differentials and truncated differences

An *impossible differential* is a pair of differences $(\Delta_{\text{in}}, \Delta_{\text{out}})$ such that no input pair with difference Δ_{in} can produce output difference Δ_{out} [BBS99, Knu98]. The usual way to find one is the *miss-in-the-middle* approach, where a difference is propagated deterministically forward from the input and backward from the output, and one looks for a position where the two propagations contradict each other. To run such a search automatically, every operation of the cipher needs a deterministic truncated model, and Subsection 4.2 provides one for modular multiplication.

The model is *truncated* in the sense of Knudsen [Knu94], that is, instead of keeping the exact value of every difference bit, it keeps only a coarser three-state description. It is *deterministic* in the sense that a bit is pinned to a value only when every consistent input forces it, and otherwise it is left unknown. A difference bit therefore takes one of three states, 0 (inactive, certainly 0), 1 (active, certainly 1), and ? (unknown). In the bitwise SAT model each bit is represented by two Boolean variables which we write as a pair $(\cdot_0, \cdot_1)$, where the component $\cdot_0$ flags that the bit is unknown and the component $\cdot_1$ flags that the bit is a determined one, so the three states are $0 = (0,0)$, $1 = (0,1)$ and $? = (1,0)$. It is to be noted that these two subscripts are only labels of the two variables of a single bit, and they do not indicate significance, unlike the index i in x_i, which does. In the MILP model, a bit is instead a single integer variable $x^{\text{c}} \in \{0,1,2\}$ with 2 denoting ?.

A component model is sound for impossible differential search when it (i) permits every real transition, so it never forbids an actual trail, and (ii) marks an output bit determined only when it is determined for every input, so it never asserts a false 0 or 1. Under (i) and (ii) a contradiction in the middle is genuine so the the differential is impossible. We call a model *exact* when, in addition, no sound truncated model can determine more, so the component discards nothing that the truncated abstraction could keep.

3 The MSX block cipher

MSX is a family of lightweight block ciphers proposed by Minematsu *et al.* [MSR+26]. Most lightweight designs build nonlinearity from small S-boxes or from addition-rotation-XOR operations. MSX instead draws its nonlinearity from modular addition and *modular multiplication* on 32-bit words. The family has two members, MSX-64 with a 64-bit block, a 128-bit key and 14 rounds, and MSX-128 with a 128-bit block, a 128- or 256-bit key and 18 rounds. We write MSX-128-256 for the 256-bit key version. We follow the word indexing of Algorithm 2 of [MSR+26].

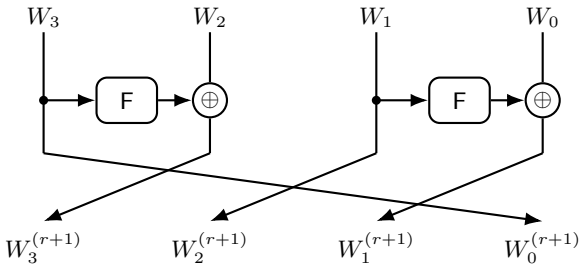

Figure 1: One round of MSX-128 as a Type-2 generalized Feistel network. The round function F is applied to W_1 and W_3 and added onto W_0 and W_2, after which the four words are cyclically permuted. All nonlinearity lies inside F (Figure 2).

State and round structure. MSX-128 is a Type-2 generalized Feistel network on four 32-bit words. The 128-bit state is written $M = W_3 \,\|\, W_2 \,\|\, W_1 \,\|\, W_0$, so in the bit numbering of our tables the input bits $32i$ to $32i + 31$ hold W_i, read little-endian as in the reference code, and the output bits 0–31, 32–63, 64–95, 96–127 hold W_1, W_2, W_3, W_0. Each of the 18 rounds applies the keyed round function F to the two words W_1 and W_3, adds the outputs onto W_0 and W_2, and cyclically permutes the four words (Figure 1). Writing $W_i^{(r)}$ for word i after r rounds, the state map is

$$\begin{aligned} W_3^{(r+1)} &= W_2^{(r)} \oplus \mathsf{F}(W_3^{(r)}), & W_2^{(r+1)} &= W_1^{(r)}, \\ W_1^{(r+1)} &= W_0^{(r)} \oplus \mathsf{F}(W_1^{(r)}), & W_0^{(r+1)} &= W_3^{(r)}. \end{aligned} \tag{1}$$

MSX-64 is the two-branch balanced Feistel obtained from the same construction on two words. Its state is written $M = W_1 \,\|\, W_0$, so in the same bit numbering the input bits 0–31, 32–63 hold W_0, W_1 and the output bits hold W_1, W_0. The word permutation provides the diffusion between branches, while all the nonlinearity is confined to F.

The round function. The round function $\mathsf{F}\colon \mathbb{F}_2^{32} \to \mathbb{F}_2^{32}$ takes a 32-bit word X, six 32-bit round-key words $\mathsf{rk}[0], \ldots, \mathsf{rk}[5]$, and a round constant c (Figure 2). Let lsb and msb denote the low and the high 16 bits of X, each read as a 32-bit word by prepending sixteen zeros, so that the additions and the products below are on 32-bit words. Let $\boxplus$ and $\otimes$ denote addition and multiplication modulo 2^{32}, and let $\lll$ denote a left rotation. It computes two products,

$$w_0 = \big((\mathrm{lsb} \boxplus \mathsf{rk}[0]) \otimes (\mathrm{msb} \boxplus \mathsf{rk}[1])\big) \boxplus \mathsf{rk}[2], \;\; w_1 = \Big(\big((\mathrm{lsb} \boxplus \mathsf{rk}[3]) \otimes (\mathrm{msb} \boxplus \mathsf{rk}[4])\big) \boxplus \mathsf{rk}[5]\Big) \oplus c,$$

keeps the high 16 bits of each, $y = \mathrm{msb}_{16}(w_1) \,\|\, \mathrm{msb}_{16}(w_0)$, and applies a linear mixing

$$\mathsf{F}(X) = y \oplus (y \lll 13) \oplus (y \lll 21).$$

Here, as in the reference code, c carries the 16-bit round constant in its high half, $c = RC \cdot 2^{16}$, so the XOR before the truncation has the same effect as XORing RC into $\mathrm{msb}_{16}(w_1)$ afterward. The nonlinear operations of F are the modular additions $\boxplus$, which mask the operands and combine the products, and the two modular multiplications $\otimes$, while the 16-bit truncation and the rotations are linear. Modular addition already has an exact model for each of the two properties we study, so among the operations of MSX, only modular multiplication lacks such a model.

Key schedule. We describe the schedule of MSX-128-256, since that is the version the key recovery of Section 7 attacks. Its 256-bit master key is eight 32-bit words $k_0, \ldots, k_7$. The

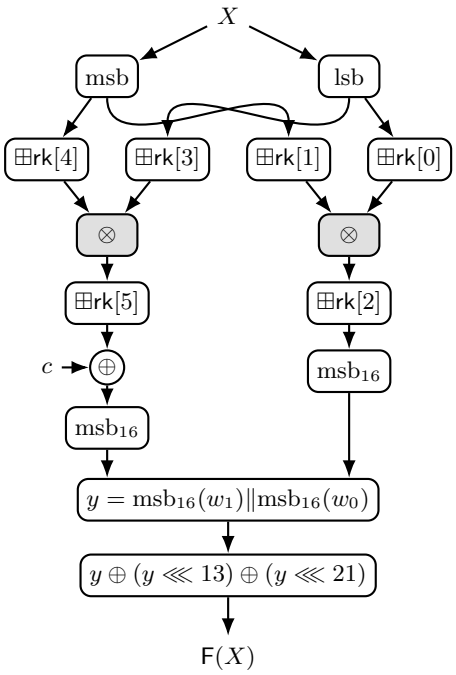

Figure 2: The MSX round function F. The two shaded boxes are modular multiplications modulo 2^{32}, the operation modeled exactly in Subsection 4.1; together with the modular additions $\boxplus$ (addition modulo 2^{32}) they are the nonlinear operations of F, while msb_{16}, which keeps the high 16 bits, and the left rotations $\lll$ are linear.

schedule of [MSR+26] (Algorithm 5) expands them into a 24-word state $S = (S_0, \ldots, S_{23})$ from which the round keys are read. The initial state $S^{(0)}$ masks each key word by a key-schedule constant, so for $i \geq 2$ we have $S_i^{(0)} = k_{\varphi(i)} \oplus KC_i$ with φ a fixed index map, while the first two words mix four key words each,

$$S_0^{(0)} = (k_1 \boxplus k_2 \boxplus k_4 \boxplus k_5) \oplus KC_0, \qquad S_1^{(0)} = (k_0 \boxplus k_3 \boxplus k_6 \boxplus k_7) \oplus KC_1. \tag{2}$$

The state is then advanced by update steps: step m takes S_m as source and overwrites nine target words by

$$S_{\mathrm{dst}} \leftarrow \mathrm{ROL}(S_{\mathrm{dst}}, \rho) \boxplus S_m, \qquad \rho \in \{1, \ldots, 9\}, \tag{3}$$

leaving S_m unchanged; we write $S^{(m)}$ for the state after m steps. Before the update step m, the state supplies the round keys of rounds $2m+1$ and $2m+2$, read as four six-word groups. Round $2m+1$ takes $S_0, \ldots, S_5$ and $S_6, \ldots, S_{11}$, and round $2m+2$ takes $S_{12}, \ldots, S_{17}$ and $S_{18}, \ldots, S_{23}$, one group per F call. So a single step supplies two consecutive rounds; the state advances once every two rounds, and the initial state, together with the eight steps, covers all 18 rounds. As required by the 3SDP-woU setting, we treat the round function and the key schedule together as a single public map, so the round-key words serve as input variables for the monomial prediction model. The key recovery of Section 7 inverts this schedule to pass from recovered round keys back to the master key. The other two variants follow the same pattern. MSX-128-128 uses the same 24-word state and the same nine update targets, and differs in the initialization, where four master-key words fill the state and only S_0 mixes several of them. MSX-64 uses a 12-word state with five update targets per step and six steps, and since its round applies F once, a step supplies one six-word group per round.

We treat both members of the family, and the main analysis is on MSX-128, round reduced to r of its 18 rounds, with the key recoveries on MSX-128-256. We work in the single-key, chosen-plaintext model. *Data* is the number of chosen plaintexts an attack queries, *time* is the number of r-round MSX evaluations, or of round function evaluations where these dominate, and *memory* is counted in bits. A distinguisher is meaningful when its data stays below the 2^{128} codebook, and a key recovery is meaningful when its time stays below the 2^{256} cost of exhaustive key search and its data below 2^{128}.

4 Modeling modular multiplication

Modular multiplication is the single operation that keeps MSX outside the scope of automated analysis. It is the only nonlinear operation of `F` without an exact model (Section 3), since the modular additions, which are also nonlinear, already have one for both of the properties we study. We give the two missing models here.

The two models are not the same, as the two properties are not the same. Monomial prediction asks which monomials appear in the ANF, so its model must preserve the parity of the number of trails, and it works on exponent vectors. Impossible-differential search asks which difference bits are forced, so its model must be deterministic, and it works on the three-state truncated encoding of Subsection 2.6. Both are exact for the multiplication, and hence neither is the loose step in a search.

4.1 A monomial-prediction model of modular multiplication

We model $z = (x \cdot y) \bmod 2^n$ exactly at the bit level. Expanding the product turns it into partial products that are accumulated by modular additions, and at every step, an `AND` gate, a `COPY`, a wire shift, or an addition has an exact 3SDP-woU rule. The model uses no truth table and maintains exact parity of 0 or 1 for every monomial, so it applies directly to the 32-bit multiplication in MSX and supports every monomial-prediction task.

In more detail, expressing both operands in their base-2 representations $x = \sum_{i=0}^{n-1} x_i 2^i$ and $y = \sum_{j=0}^{n-1} y_j 2^j$, the n-bit modular multiplication $z = (x \cdot y) \bmod 2^n$ can be expanded as

$$z = \left(\sum_{i=0}^{n-1} x_i 2^i\right)\left(\sum_{j=0}^{n-1} y_j 2^j\right) \bmod 2^n = \sum_{j=0}^{n-1}\left(\sum_{i=0}^{n-1}(x_i \cdot y_j)2^{i+j}\right) \bmod 2^n.$$

This decomposition reduces modular multiplication to a sequence of basic operations on individual bits. The algorithm iterates over each multiplier bit y_j. For each y_j, it computes the scalar products $x_0 y_j, \ldots, x_{n-1} y_j$, each modeled as an exact bitwise `AND` on exponent variables, with COPY constraints handling the multiple uses of x_i and y_j across partial products. The factor 2^{i+j} dictates a left-shift by j positions; bits where $i + j > n - 1$ are truncated. The remaining bits form a shifted word s, which is added to the running output z (initialized to 0) with an exact modular addition model that follows the perfect monomial prediction of Hu and Yap [HY24]. Iterating over all j accumulates the partial products into the final output $z = (z_0, \ldots, z_{n-1})$, as detailed in Algorithm 1.

Exactness. The feasible trails of the model are in one-to-one correspondence with the 3SDP-woU trails of $z = (x \cdot y) \bmod 2^n$, with the correct multiplicity, so by the criterion of Subsection 2.3 the model gives every monomial the same parity as the multiplication itself. Each partial product $p_{i,j} = x_i y_j$ is an exact `AND`, the repeated uses of x_i and y_j are split by `COPY` (as in Subsection 2.5), the shift only rewires exponent positions, and the accumulation uses the exact modular-addition model, whose carry exponents are uniquely determined for each trail [HY24]. Since every step is exact, so is their composition, and the model is exact for every n.

Algorithm 1: MILP constraints for modular multiplication

Input: MILP model $\mathcal{M}$, inputs $x = (x_i)_{i=0}^{n-1}$ and $y = (y_j)_{j=0}^{n-1}$

Create output word $z = (z_0, \ldots, z_{n-1})$ and initialize to 0

Create partial products $p_{i,j} = x_i y_j$ for all i, j with $i + j < n$ (bits with $i + j \geq n$ are truncated)

// The constraints below make $p_{i,j}$ active only when both x_i and y_j are (the AND $x_i y_j$), and let each x_i and y_j distribute over its partial products (the COPY rule, cf. Subsection 2.5).

for $i = 0$ **to** $n - 1$ **do**
  for each j with $i + j < n$ **do**
    $\mathcal{M}$.addConstr($x_i \geq p_{i,j}$)
  $\mathcal{M}$.addConstr$\left(\sum_{j:\, i+j<n} p_{i,j} \geq x_i\right)$

for $j = 0$ **to** $n - 1$ **do**
  for each i with $i + j < n$ **do**
    $\mathcal{M}$.addConstr($y_j \geq p_{i,j}$)
  $\mathcal{M}$.addConstr$\left(\sum_{i:\, i+j<n} p_{i,j} \geq y_j\right)$

for $j = 0$ **to** $n - 1$ **do**
  Create shifted word $s = (s_0, \ldots, s_{n-1})$
  for $k = 0$ **to** $n - 1$ **do**
    if $k < j$ **then**
      $\mathcal{M}$.addConstr($s_k = 0$)
    else
      $\mathcal{M}$.addConstr($s_k = p_{k-j,\, j}$)
  $z \leftarrow$ modular_addition($\mathcal{M}, z, s$)

We have implemented the model in CLAASP-MP [BRT26]. To confirm that the implementation realises the model above, we evaluated for small n the ANF it produces at all input pairs (x, y) and found it equal to the integer product $(x \cdot y) \bmod 2^n$.

4.2 A deterministic truncated model of modular multiplication

We now give the second model, the one an impossible differential search needs. For modular addition such a model is well known and it is built on the carry ripple pivot of an adder. For modular multiplication we give one here, at the bit level, and we show that it is not only sound but exact for this operation. Consequently every incompatibility that a miss-in-the-middle search reports through a multiplication is genuine, and the multiplication step itself does not discard anything that the truncated abstraction could keep. Throughout we consider two independent operands with modulus 2^n, which is the setting used in MSX and is also the standard convention for a modular multiplication. The three-state encoding and the two soundness obligations are those of Subsection 2.6.

Fix the bit conventions: $x = \sum_{i=0}^{n-1} x_i 2^i$ and $y = \sum_{j=0}^{n-1} y_j 2^j$, with x_0, y_0 the least significant bits and x_{n-1}, y_{n-1} the most significant, and the output $z = (xy) \bmod 2^n = \sum_{k=0}^{n-1} z_k 2^k$ written the same way. A truncated pattern fixes each $\Delta x_i, \Delta y_j \in \{0, 1, ?\}$, and we call a bit *active or unknown* when it is 1 or ?. Let p be the least index at which either operand has an active or unknown bit, and $p = n$ when $\Delta x = \Delta y = 0$.

Lemma 1. *For every m, $(xy) \bmod 2^m$ depends only on $x \bmod 2^m$ and $y \bmod 2^m$.*

Proof. Write $x = x' + 2^m \lfloor x/2^m \rfloor$ and $y = y' + 2^m \lfloor y/2^m \rfloor$ with $x' = x \bmod 2^m$ and $y' = y \bmod 2^m$. Expanding xy, every term other than $x'y'$ carries a factor 2^m and vanishes modulo 2^m. Hence $(xy) \bmod 2^m = (x'y') \bmod 2^m$. □

Proposition 1. *Let x and y be independent operands, each ranging over all n-bit values. Then $\Delta z_k = 0$ for every $k < p$, and for every $k \geq p$ the bit Δz_k is not fixed by the truncated pattern; in particular no output bit is ever forced to* 1.

Proof. Case $k < p$. By the choice of p, the bits $\Delta x_0, \ldots, \Delta x_{p-1}$ and $\Delta y_0, \ldots, \Delta y_{p-1}$ are all 0, so x and $x \oplus \Delta x$ agree on their low p bits, that is $x \equiv x \oplus \Delta x \pmod{2^p}$, and likewise $y \equiv y \oplus \Delta y \pmod{2^p}$. By Lemma 1 with $m = p$, $(xy) \bmod 2^p = \big((x \oplus \Delta x)(y \oplus \Delta y)\big) \bmod 2^p$, so z and its differenced value agree on bits $0, \ldots, p-1$; hence $\Delta z_k = 0$ for all $k < p$.

Case $k \geq p$. Recall that p is the lowest active or unknown position, so the difference is zero below p, while any number of bits at or above p may be active or unknown. Say the bit at position p is active or unknown in x. If instead it is so in y, then we exchange the roles of x and y throughout. We keep the difference $(\Delta x, \Delta y)$ fixed, any admissible one with $\Delta x_p = 1$, and we vary only the base values x and y, which the pattern leaves free. We first show that the output difference 0 is attainable. With $x = 0$ and $y = \Delta y$ we have $z = xy = 0$ and $y \oplus \Delta y = 0$, so $z' = (x \oplus \Delta x)(y \oplus \Delta y) = \Delta x \cdot 0 = 0$ and $\Delta z = 0$ on every bit. Hence no output bit is ever forced to 1. We next show that the value 1 is attainable at each $k \geq p$. With $x = 0$ and $y = 2^{k-p} \oplus \Delta y$ we have $y \oplus \Delta y = 2^{k-p}$, so $z = 0$ and $z' = (x \oplus \Delta x)(y \oplus \Delta y) = \Delta x \cdot 2^{k-p}$, which is Δx shifted up by $k - p$ places. Its bit k equals $\Delta x_p = 1$, and hence $\Delta z_k = 1$. The base $y = 2^{k-p} \oplus \Delta y$ absorbs whatever Δy is, so this needs no condition on the difference and no single active bit. Thus for every $k \geq p$, including $k = p$ itself, the bit Δz_k takes both 0 and 1 and is not fixed. □

The determined region therefore stops exactly at p, which we call the *pivot*. Every bit below the pivot is a determined zero, every bit from the pivot upward is unknown, and no bit is ever a determined one. It is to be noted that both the obligations of Subsection 2.6 are met. For (ii), the model marks a bit as determined only below p, where Proposition 1 shows that it is genuinely zero for every input. For (i), it leaves every other bit unknown, which permits any real transition. For independent operands, no sound truncated model can determine more, and hence the rule is not merely sound but exact.

Remark 1 (independent operands). In the proof we choose the base values of x and y independently, by setting $x = 0$ and then moving y. If a single word feeds both the operands, so that the component computes $x \mapsto x^2 \bmod 2^n$, then the input space is restricted to the diagonal and more bits become determined. Writing $\Delta x = 2^p$, the XOR flips the bit at position p, so $x \oplus 2^p$ equals $x + 2^p$ when $x_p = 0$ and equals $x - 2^p$ when $x_p = 1$. In either case $(x \pm 2^p)^2 - x^2 = \pm 2^{p+1}x + 2^{2p}$, which is divisible by 2^{p+1} whenever $p \geq 1$. Hence the two squares agree modulo 2^{p+1}, and the determined zero-run reaches through bit p itself, that is one place further than in the general case. For $p = 0$ the difference is $\pm 2x + 1$, which is always odd, and hence Δz_0 is a determined 1. For example, at $n = 4$ and $\Delta x = 2^0, 2^1, 2^2$, the true patterns are 1 0 ? ?, 0 0 ? ? and 0 0 0 ? with the least significant bit first, whereas the model gives ? ? ? ?, 0 ? ? ? and 0 0 ? ?. The model therefore stays sound, because a bit which is constant for all pairs (x, y) is in particular constant when $y = x$, but it is no longer exact. The search may then miss an incompatibility, but it never reports a false one. This does not arise in MSX, where the two operands come from distinct halves of a state word.

Constraints for the SAT and MILP models. The same rule can also be stated position by position, and this is the form in which the constraints are written. For each position i, let the activity flag $a_i = 1$ when the input difference is active or unknown there, that is $\Delta x_i \in \{1, ?\}$ or $\Delta y_i \in \{1, ?\}$, and $a_i = 0$ otherwise. Accumulate these flags from the least significant bit up to position k into the *accumulated activity flag* A_k, a running OR that switches on at the lowest active or unknown position and then stays on,

$$A_{-1} = 0, \qquad A_k = A_{k-1} \vee a_k = a_0 \vee a_1 \vee \cdots \vee a_k.$$

Then $A_k = 1$ exactly when $k \geq p$, and hence Δz_k is unknown when $A_k = 1$ and $\Delta z_k = 0$ when $A_k = 0$. In the integer encoding the rule sets Δz_k to $2A_k$, and in the bitwise encoding it sets $z_{k,0} = A_k$ and $z_{k,1} = 0$. For instance at $n = 4$, with a single active bit at position 2 in one operand, the flags are $A = (0, 0, 1, 1)$ and the output pattern is $0\,0\,?\,?$.

Algorithm 2 gives the MILP constraints and Algorithm 3 the SAT clauses.

Algorithm 2: MILP constraints for the deterministic truncated modular multiplication

Input: MILP model $\mathcal{M}$; operand differences $x = (x_0, \ldots, x_{n-1})$ and $y = (y_0, \ldots, y_{n-1})$ and output difference $z = (z_0, \ldots, z_{n-1})$, each x_j, y_j, z_j an integer variable in $\{0, 1, 2\}$ (0 inactive, 1 active, 2 unknown), with bit 0 the least significant

// a_j^x, a_j^y `flag an active or unknown bit (≥ 1) in each operand`

for $j = 0$ **to** $n - 1$ **do**
  Create binaries a_j^x with $a_j^x = 1 \iff x_j \geq 1$, and a_j^y with $a_j^y = 1 \iff y_j \geq 1$ Create binary a_j
  $\mathcal{M}$.addConstr($a_j \geq a_j^x$)
  $\mathcal{M}$.addConstr($a_j \geq a_j^y$)
  $\mathcal{M}$.addConstr($a_j \leq a_j^x + a_j^y$)

// $A_j = a_0 \vee \cdots \vee a_j$`, accumulated from the least significant bit`

Create binary A_0
$\mathcal{M}$.addConstr($A_0 = a_0$)
for $j = 1$ **to** $n - 1$ **do**
  Create binary A_j
  $\mathcal{M}$.addConstr($A_j \geq a_j$)
  $\mathcal{M}$.addConstr($A_j \geq A_{j-1}$)
  $\mathcal{M}$.addConstr($A_j \leq a_j + A_{j-1}$)
for $j = 0$ **to** $n - 1$ **do**
  $\mathcal{M}$.addConstr($z_j = 2\,A_j$)

The bitwise SAT model expresses the same rule directly on the pair encoding, with the accumulated flag A_j carried by the $\cdot_0$ component of the output. Each equivalence $z_{j,0} \Leftrightarrow (\ell_1 \vee \cdots \vee \ell_m)$ expands to the clauses $(\neg\ell_i \vee z_{j,0})$ for every literal ℓ_i together with the single clause $(\neg z_{j,0} \vee \ell_1 \vee \cdots \vee \ell_m)$, and each unit clause fixes $z_{j,1}$ to 0 so that no output bit is ever a determined 1.

Algorithm 3: SAT clauses for the deterministic truncated modular multiplication

Input: clause set $\mathcal{C}$; operand differences as bit pairs $(x_{j,0}, x_{j,1})$ and $(y_{j,0}, y_{j,1})$ and output pairs $(z_{j,0}, z_{j,1})$, $j = 0, \ldots, n-1$, with bit 0 the least significant; the component $\cdot_0$ flags unknown and $\cdot_1$ flags a determined one, so $0 = (0, 0)$, $1 = (0, 1)$, $? = (1, 0)$

// $z_{j,0}$ `carries the accumulated flag` A_j

Add to $\mathcal{C}$ the clauses of $z_{0,0} \Leftrightarrow (x_{0,0} \vee x_{0,1} \vee y_{0,0} \vee y_{0,1})$
Add to $\mathcal{C}$ the unit clause $(\neg z_{0,1})$
for $j = 1$ **to** $n - 1$ **do**
  Add to $\mathcal{C}$ the clauses of $z_{j,0} \Leftrightarrow (x_{j,0} \vee x_{j,1} \vee y_{j,0} \vee y_{j,1} \vee z_{j-1,0})$
  Add to $\mathcal{C}$ the unit clause $(\neg z_{j,1})$

We have implemented the model in CLAASP, both for SAT and for MILP, as the deterministic truncated constraints of the modular multiplication component. To confirm that the implementation realises the rule of Proposition 1, we propagated every input pattern for small word sizes and compared the result against the definition $(xy) \bmod 2^n$, and we also confirmed that the SAT and the MILP builds agree with each other on MSX.

5 Algebraic distinguishers of MSX

We now apply the monomial-prediction model of Subsection 4.1, through CLAASP-MP, to MSX. The component model is realised inside the CLAASP-MP monomial-prediction framework [BRT26], built on the CLAASP library [BGG+23], which assembles a single global MILP $\mathcal{M}$ from the directed acyclic graph that CLAASP uses to represent a primitive. Each edge of the graph carries an exponent variable, each component contributes the linear constraints of its 3SDP-woU rule, and the exponent variables attached to a wire are created once and reused by every component that reads it. Hence $\mathcal{M}$ captures the complete algebraic structure of the primitive, and its feasible 0/1 solutions are exactly the monomial trails. For MSX the one component this assembly lacked is the modular multiplication of Subsection 4.1, and with it in place the cipher and its key schedule together form a single model. Every result below is a query on this one fixed model, obtained by adding a few task-dependent constraints and an objective, and we use two such queries. Both are available as methods of CLAASP-MP, so the same bounds can be computed for any primitive already described in CLAASP without setting up the constraints by hand.

All MILP instances are solved with Gurobi. Every degree computation here needs only one optimal solution, so we run them with `PoolSearchMode` $= 0$ and a zero MIP gap, which makes the solver prove optimality rather than stop within a tolerance. For the degrees restricted to a cube we additionally set `OptimalityTol` $= 10^{-9}$, which guards against a false optimum arising from floating-point inaccuracy. The experiments were run on an Apple M1 Max with 32 GB of memory, and the bottleneck is solver time rather than memory.

We treat both variants and focus on MSX-128. Every degree and balance value reported below is an output of CLAASP-MP under the model of Subsection 4.1.

Degree bound. To bound the algebraic degree of an output bit in a chosen set of input variables, we fix the exponent variable of that output bit to one, and we maximize the number of active input exponent variables by adding a linear objective that maximizes their sum over the chosen set. The optimum d is an upper bound on the algebraic degree in those variables.

Table 2 reports these bounds for both variants of MSX over the first seven rounds. For brevity, bounds are grouped by output branch, as all bits within a branch share the same degree growth pattern.

Table 2: Upper bounds on the algebraic degree of the MSX cipher.

Variant	Output Bits	Rounds						
		1	2	3	4	5	6	7
MSX-64	0–31	32	63	63	63	63	63	63
	32–63	1	32	63	63	63	63	63
MSX-128	0–31, 64–95	32	63	94	125	126	126	127
	32–63, 96–127	1	32	63	94	125	126	126

As shown in Table 2, the algebraic degree upper bounds of MSX-64 saturate by round 3. For MSX-128, CLAASP-MP reports a degree upper bound of 126 at round 6 for all 128 output bits, which already yields an integral distinguisher of data complexity 2^{127}. The table also exhibits a regular one-round lag between its two rows, which is enforced by the copy branches of the Type-2 structure.

Since W_0, W_2 never enter F, the round map (1) makes them verbatim copies of the previous F-input words, $W_0^{(r)} = W_3^{(r-1)}$ and $W_2^{(r)} = W_1^{(r-1)}$. As functions of the plaintext and key, they thus have the same ANF as $W_3^{(r-1)}, W_1^{(r-1)}$, so their cube sums and their degree in any set of variables agree with those of W_3, W_1 one round earlier. This has two consequences. First, any cube that leaves the F-input words W_3, W_1 balanced after

$r-1$ rounds leaves the copy words W_0, W_2 balanced after r rounds, at the same data cost. Second, the degree of W_0, W_2 trails that of W_3, W_1 by one round, which is why the $\{32\text{–}63, 96\text{–}127\}$ row of Table 2 equals the $\{0\text{–}31, 64\text{–}95\}$ row shifted by one round and why its last entry is 126 (as $W_0^{(7)} = W_3^{(6)}$). So a degree-based distinguisher on W_0, W_2 survives to round 7 with 2^{127} data, which is one round beyond the round-6 bound for all bits, and monomial prediction then gives a cube distinguisher with much less data.

Balance test for a cube. We fix a cube $\mathcal{C}_I$ as in Subsection 2.4, by selecting a set I of plaintext bits as cube variables and fixing the remaining plaintext bits to zero. We add these constraints to the model and maximize, for each output bit, the degree D over the cube variables. If $D < |I|$ then the monomial of degree $|I|$ is absent from the ANF of that output bit, so the bit sums to zero over all $2^{|I|}$ cube inputs and is balanced. The key recoveries of Section 7 also use other values of the fixed bits, so for their cubes we repeat the test with these bits left free, and the degree still stays below $|I|$. Hence no monomial of the ANF is divisible by $\boldsymbol{x}_I$, and the sum vanishes for every choice of the fixed bits. A balanced set of output bits obtained this way is a deterministic integral distinguisher, since it holds for every key. For MSX-64, any 2-dimensional cube from W_0 leaves all bits in W_0 balanced at round 2, and a 33-dimensional cube over bits $\{0\text{–}32\}$ leaves W_0 balanced at round 3, which is consistent with the saturation bounds of Table 2.

For MSX-128, Table 3 summarizes the monomial prediction results. Across early rounds, the balanced outputs of low-dimensional cubes gradually narrow, reflecting progressive yet incomplete inter-word mixing.

Table 3: Monomial prediction in MSX-128: active bits I, cube dimension $d = |I|$, maximum degree D in cube variables, and balanced output words W_j.

Rounds	**3**	**4**		**5**	**6**	**7**	**8**
Cube Bits (I)	$\{0,1\}$	$\{0,1\}$	$\{0-31\}$	$\{0-32\}$	$\{0-63\}$	$\overline{\{32-63\}}$	$\overline{\{32\}}$
Dim (d)	2	2	32	33	64	96	127
Max deg (D)	1	1	1	32	63	95	126
Balanced Words	W_3, W_0	W_0	W_0	W_2, W_0	W_2	W_2, W_0	W_2, W_0

The balance pattern is word-aligned, so for a cube confined to a single input word, which output words become balanced depends only on that word and not on the chosen bit positions within it. For any 2-dimensional cube taken from one word, the balanced outputs are therefore fixed by the source word alone:

- *Round* 3. A cube in W_0 leaves W_3, W_0 balanced, one in W_1 leaves W_0, one in W_2 leaves W_1, W_2, and one in W_3 leaves W_2.
- *Round* 4. A cube in W_0 leaves W_0 balanced and one in W_2 leaves W_2, while cubes in W_1 and in W_3 no longer leave any word balanced.
- *Round* 5. No cube leaves any word balanced, whichever word it is taken from.

Low-dimensional cubes narrow quickly, balancing a few output words at round 3 and none by round 5, so the distinguisher has to come from a higher dimension. At round 6 the active set $W_0 \cup W_1$ with $d = 64$ leaves W_2 balanced, and $W_2 \cup W_3$ with $d = 64$ leaves W_0 balanced.

The last two columns of Table 3 give the two distinguishers, and the copy structure above explains them. The 96-dimensional cube over $W_0 \cup W_2 \cup W_3$ leaves the F-input words W_3, W_1 balanced at round 6: their degree in the 96 cube variables is $95 < 96$, so the degree-96 monomial cancels, and the copy structure carries this to W_0, W_2 at round 7 at the same cost, which is exactly the round-7 entry the CLAASP-MP reports. The symmetric cube over $W_0 \cup W_1 \cup W_2$, obtained from the first by the word-aligned symmetry of MSX-128, gives a second, independent witness. This is the 7-round deterministic integral distinguisher of Table 1: a 96-bit cube leaves 64 of the 128 output bits (W_0 and W_2) balanced, with data complexity 2^{96}. The last column comes from a different cube. Fixing

one input bit in W_1 or W_3 leaves 127 active bits, and in this cube W_3, W_1 are balanced at round 7, since their restricted degree is $126 < 127$. The copy structure carries this to W_0, W_2 at round 8, which is the round-8 entry of Table 3. This 8-round distinguisher costs 2^{127} and drives the 9-round key recovery of Section 7.

Both of these distinguishers are used in Section 7. Adding one round to the 7-round distinguisher gives a key recovery on 8 rounds, and adding one round to the 8-round distinguisher gives one on 9 rounds.

6 Impossible differentials of MSX

For impossible differentials the designers follow the approach of the Midori design [BBI+15] and report that the maximum number of rounds for which an impossible differential characteristic exists is estimated to be seven for MSX-64 and eleven for MSX-128. We describe this number precisely.

The estimate rests on one quantity. Under the standard truncated assumption that the round function diffuses completely, that is $\mathsf{F}(\text{active}) = \text{active}$ and $\mathsf{F}(0) = 0$, they report that there is no probability one truncated differential trail for three rounds of MSX-64 and five rounds of MSX-128. A miss-in-the-middle search then gives an estimate of seven and eleven rounds. The estimate is therefore an upper bound rather than a construction, and it uses no model of the modular multiplication. The designers use it to argue their security margin. The same pattern appears in Midori, whose approach they follow. There the same count gives an estimate of seven rounds, while the impossible differential the Midori designers actually construct is six rounds long, so the estimate sits one round above the construction. It does leave open what a search that models the multiplication would find.

The search model. We run the miss-in-the-middle search of Subsection 2.6 fully automatically. The model is assembled from the same CLAASP graph as in Section 5, with each component now contributing its deterministic truncated constraints instead of its 3SDP-woU ones, and with the cipher instantiated twice, once forward from the plaintext difference and once backward from the ciphertext difference, so that the two propagations meet at every round output. A pair of differences is impossible exactly when the model is satisfiable, since a satisfying assignment gives the forward and the backward propagation that contradict each other. These searches use the SAT backend of CLAASP with the solver Kissat, and they were run on the same machine as the monomial-prediction experiments.

Single-bit search. We fixed a single active plaintext bit and a single active ciphertext bit and ran through all $128 \times 128 = 16384$ such pairs at each round count, so the sweep is exhaustive at this granularity. Table 4 reports how many of them are impossible.

The count falls monotonically with the round count and it reaches zero at eight rounds, so the maximum reached by this search is seven rounds. There are 1984 of them at seven rounds, and each is explicit, given as a pair of one active plaintext bit and one active ciphertext bit, with the forward and backward propagation available from the model for any pair. We also went beyond single-bit inputs at eight rounds, where the count first reaches zero. We searched 62596 pairs with two active input bits and one active output bit, drawn at random from the 1040384 such pairs. A separate run covered a further 12192 pairs with two or three active input bits and one or two active output bits. We found no impossible differential in any of them.

Table 4: Single-bit impossible differentials of MSX-128-256 in the single-key setting. All $128 \times 128 = 16384$ pairs of one active plaintext bit and one active ciphertext bit were searched at each round count.

rounds	pairs searched	not detected as impossible	impossible
5	16384	2176	14208
6	16384	8320	8064
7	16384	14400	1984
8	16384	16384	0
9	16384	16384	0
10	16384	16384	0
11	16384	16384	0

7 Key-recovery attacks on MSX-128-256

The 7-round distinguisher of Section 5 extends to a key-recovery attack on 8-round MSX-128 with a 256-bit key, and the 8-round distinguisher extends in the same way to 9 rounds. Recall from Section 3 that each round applies the round function F to W_1 and W_3 and permutes the four words; for the key recovery we make the round keys explicit. Here $\mathsf{F}\colon \mathbb{F}_2^{32} \times \mathbb{F}_2^{192} \to \mathbb{F}_2^{32}$, $(w, \mathsf{rk}) \mapsto \mathsf{F}(w; \mathsf{rk})$, and a subkey group $\mathsf{rk} = (\mathsf{rk}[0], \ldots, \mathsf{rk}[5]) \in (\mathbb{F}_2^{32})^6$ is six 32-bit words; each round uses two groups $\mathsf{rk}_{r,0}, \mathsf{rk}_{r,1}$. Writing $W_i^{(r)}$ for word i after r rounds, the state map is

$$\begin{aligned} W_3^{(r+1)} &= W_2^{(r)} \oplus \mathsf{F}\big(W_3^{(r)}; \mathsf{rk}_{r,1}\big), & W_2^{(r+1)} &= W_1^{(r)}, \\ W_1^{(r+1)} &= W_0^{(r)} \oplus \mathsf{F}\big(W_1^{(r)}; \mathsf{rk}_{r,0}\big), & W_0^{(r+1)} &= W_3^{(r)}, \end{aligned} \tag{4}$$

so each round consumes two subkey groups $\mathsf{rk}_{r,0}, \mathsf{rk}_{r,1}$ (384 bits in total). Only W_1 and W_3 enter F; the words W_0, W_2 are verbatim copies of the previous W_3, W_1. The 7-round distinguisher (Table 3) balances exactly these two copy words W_0, W_2, and because of this copy structure one round is cheap to remove.

Inverting the last round. Given the 8-round state $\big(W_3^{(8)}, W_2^{(8)}, W_1^{(8)}, W_0^{(8)}\big)$, the copy rules of (4) at $r = 7$ read the two 7-round F inputs directly, $W_3^{(7)} = W_0^{(8)}$ and $W_1^{(7)} = W_2^{(8)}$. Substituting these into the two nonlinear lines of (4) recovers the two balanced 7-round words:

$$W_0^{(7)} = W_1^{(8)} \oplus \mathsf{F}\big(W_2^{(8)}; \mathsf{rk}_a\big), \qquad W_2^{(7)} = W_3^{(8)} \oplus \mathsf{F}\big(W_0^{(8)}; \mathsf{rk}_b\big), \tag{5}$$

where $\mathsf{rk}_a, \mathsf{rk}_b$ are the two 8-round subkey groups (the $\mathsf{rk}_{7,0}, \mathsf{rk}_{7,1}$ of (4)). So once we guess its 192-bit group, each balanced word is obtained from the ciphertext by a single F-evaluation. Here $W_0^{(7)}$ depends only on rk_a, and $W_2^{(7)}$ only on rk_b, so the two groups are recovered independently.

Distillation. Fix a guess of the 192-bit group rk_a and ask whether the recovered word $W_0^{(7)} = W_1^{(8)} \oplus \mathsf{F}(W_2^{(8)}; \mathsf{rk}_a)$ is balanced over the cube. Because the cube sum is $\mathbb{F}_2$-linear, it splits into a key-independent and a key-dependent part,

$$\bigoplus_{\text{cube}} W_0^{(7)} = \underbrace{\Big(\bigoplus_{\text{cube}} W_1^{(8)}\Big)}_{=:\, S} \oplus \bigoplus_{\text{cube}} \mathsf{F}(W_2^{(8)}; \mathsf{rk}_a),$$

where S is a fixed 32-bit vector read once from the data. The balance condition $\bigoplus_{\text{cube}} W_0^{(7)} = \mathbf{0}$ is therefore equivalent to $\bigoplus_{\text{cube}} \mathsf{F}(W_2^{(8)}; \mathsf{rk}_a) = S$, which the correct rk_a satisfies and a wrong one only with probability 2^{-32}.

Evaluated naively this sum has one term per cube point, so testing all 2^{192} guesses would cost $2^{96} \cdot 2^{192}$. The saving comes from one fact that $\mathsf{F}(W_2^{(8)}; \mathsf{rk}_a)$ sees the ciphertext only through the single 32-bit word $W_2^{(8)}$. This word takes at most 2^{32} values, while the cube has 2^{96} points. So on average each value repeats about 2^{64} times, and F is evaluated on the same input again and again. In an XOR sum, an input contributing an even number of times cancels, so only the *parity* of each count matters. Recording these parities in a table

$$T[v] \;=\; \#\{\text{cube points with } W_2^{(8)} = v\} \bmod 2, \qquad v \in \mathbb{F}_2^{32},$$

collapses the key-dependent sum to $\bigoplus_{v\,:\,T[v]=1} \mathsf{F}(v; \mathsf{rk}_a)$, over at most 2^{32} distinct inputs instead of 2^{96} cube points. A single streaming pass over the 2^{96} ciphertexts, done before any key is guessed, builds T and S together, and for each ciphertext we flip the bit $T[W_2^{(8)}]$ and XOR $W_1^{(8)}$ into S. The zero-sum test then reads

$$\bigoplus_{v\,:\,T[v]=1} \mathsf{F}(v; \mathsf{rk}_a) \;=\; S \tag{6}$$

This test costs at most 2^{32} evaluations of F per guess, instead of 2^{96} and the expensive data pass is performed only once.

Filtering. A single cube gives only one 32-bit test (6) on the 192-bit group rk_a. The correct rk_a always passes this test, whereas a wrong rk_a passes it only when its sum happens to equal S, that is with probability 2^{-32}. So one cube still leaves about $2^{192} \cdot 2^{-32} = 2^{160}$ wrong candidates, which are far too many. To pin the group we use six cubes, obtained by changing the value or the position of the held-constant word. Each cube gives an independent 32-bit test, so a wrong rk_a must pass all six, and this happens with probability $(2^{-32})^6 = 2^{-192}$. The expected number of surviving wrong guesses is then $2^{192} \cdot 2^{-192} = 1$, so besides the correct rk_a about one wrong guess is expected to survive, and we keep all survivors. In short, each cube removes 32 bits of uncertainty, and the six cubes together remove $6 \times 32 = 192$ bits, which is exactly the width of rk_a. We do not store the 2^{192} guesses. Instead, each guess is tested against all six tables as it is generated. Recovering rk_a then costs about $2^{192} \cdot 6 \cdot 2^{32} \approx 2^{227}$ evaluations of F. The second group rk_b is recovered in the same way from the balance of $W_2^{(7)}$, so the two groups together cost about 2^{228}.

From subkeys to the master key. It remains to recover the master key from the two 8-round groups $\mathsf{rk}_a, \mathsf{rk}_b$, which we do for every pair of surviving candidates. By the key schedule of Section 3, the round subkeys are slices of the state S, which starts at $S^{(0)}$ and advances one step every two rounds; the round-8 groups $\mathsf{rk}_a, \mathsf{rk}_b$ are therefore the twelve words $S_{12}^{(3)}, \ldots, S_{23}^{(3)}$. We recover the master key by inverting the schedule back from these.

Each step is invertible once its source is known, since ROL and $\boxplus$ are both invertible, and (3) inverts to $S_{\mathrm{dst}} = \mathrm{ROR}\big(S_{\mathrm{dst}}^{\mathrm{new}} \boxminus S_m, \rho\big)$. The schedule passes through four states, one update step apart, $S^{(0)} \xrightarrow{\text{step 0}} S^{(1)} \xrightarrow{\text{step 1}} S^{(2)} \xrightarrow{\text{step 2}} S^{(3)}$, and none of the twelve known words $S_{12}^{(3)}, \ldots, S_{23}^{(3)}$ is still in its initial form, so to undo a step on a word we need that step's source. The only source that is not directly available is the first one, $S_0^{(0)}$, a sum of four key words; we guess it, $g = S_0^{(0)}$, over its 2^{32} values, and the rest cascades:

1. *Step-0 words.* $S_{12}^{(3)}, S_{17}^{(3)}, S_{20}^{(3)}, S_{23}^{(3)}$ are touched only by step 0 (source g, rotations $\rho = 5, 7, 8, 9$). An update step rewrites only nine of the 24 state words, so a word that steps 1 and 2 both leave alone keeps the value step 0 gave it, that is $S_i^{(3)} = S_i^{(2)} = S_i^{(1)}$. One inversion of step 0 therefore carries the known word straight back to $S_i^{(0)}$. Removing the constant, $k_{\varphi(i)} = S_i^{(0)} \oplus KC_i$, yields k_0, k_6, k_3, k_7.

2. *Step*-1 *source.* Now $S_1^{(0)} = (k_0 \boxplus k_3 \boxplus k_6 \boxplus k_7) \oplus KC_1$ is known, and step 0 carries it to $S_1^{(1)} = \mathrm{ROL}(S_1^{(0)}, 1) \boxplus g$.
3. *Step*-1 *words.* $S_{13}^{(3)}, S_{15}^{(3)}$ (touched only by step 1, rotations $5, 6$) invert against $S_1^{(1)}$ to give k_1, k_4.
4. *Step*-2 *source.* $S_2^{(0)} = k_0 \oplus KC_2$ is known, and step 1 carries it to $S_2^{(2)} = \mathrm{ROL}(S_2^{(0)}, 1) \boxplus S_1^{(1)}$.
5. *Step*-2 *word.* $S_{16}^{(3)}$ (touched only by step 2, rotation 6) gives k_5.
6. *Last word.* $k_2 = (g \oplus KC_0) \boxminus k_1 \boxminus k_4 \boxminus k_5$, from the defining sum (2) of $S_0^{(0)}$.

All eight master-key words are thus a deterministic function of the single guess g. Only seven of the twelve known words enter this derivation; the remaining five, $S_{14}^{(3)}, S_{18}^{(3)}, S_{19}^{(3)}, S_{21}^{(3)}, S_{22}^{(3)}$, are $5 \times 32 = 160$ bits of redundancy, and recomputing them forward from the derived key and comparing against their known values tests the guess. We try all 2^{32} values of g for each surviving pair. A wrong g, or any g for a wrong pair, passes only with probability 2^{-160}, and a trial encryption removes any wrong survivor.

In total the 8-round attack needs $\approx 2^{99}$ chosen plaintexts (six cubes of 2^{96}), $\approx 2^{35}$ bits of memory (the parity table) and $\approx 2^{228}$ time.

Reaching nine rounds. For nine rounds we use an 8-round distinguisher. Fixing one input bit in W_1 or W_3 and varying the other 127 input bits gives a 127-dimensional cube that leaves W_0 and W_2 balanced at round 8 (their degree in the cube variables is $126 < 127$). As in the 8-round attack, we peel a single round off the 9-round output back to this round-8 property, and this recovers the round-9 subkeys. The inversion (5), the distillation, and the 2^{192} subkey guess are all the same. Only the data changes. Each cube fixes one input bit, so it is one half of the codebook, that is 2^{127} plaintexts, and we still need six of them for six independent filters. The six cubes must fix six *different* bit positions, all in W_1 and W_3. Fixing one position to 0 and then to 1 gives the same equation, because adding the two halves is the sum over the full codebook, which is always $\mathbf{0}$. Six different positions give six genuine filters. Their half-codebooks overlap heavily, and together they cover the whole codebook, except the plaintexts in which all six bits are 1. So the attack reads essentially the entire codebook once. Its cost is about 2^{128} data (a full-codebook requirement), $\approx 2^{35}$ memory and $\approx 2^{228}$ time, again for the 256-bit key only. The master key is recovered from the round-9 subkeys $S_0^{(4)}, \ldots, S_{11}^{(4)}$ by inverting the schedule in the same way. Here it needs two 32-bit source guesses instead of one, so this step costs 2^{64} per surviving pair rather than the 2^{32} of the 8-round attack. This is still negligible beside the 2^{228} subkey-recovery phase.

Key recovery from the impossible differentials. The impossible differentials of Section 6 also extend by one round. They are genuinely impossible because the model leaves the output of F unknown whenever its input is active, so the correct key is never wrongly discarded. Extending a seven-round differential by one round, therefore, gives a key recovery on eight rounds, which is one round short of what the integral attack reaches. We report it for completeness rather than as our best attack, and we do not work out its complexity.

The differential and linear bounds. The designers assess MSX against differential and linear cryptanalysis by bounding trail probabilities [MSR+26, Table 3]. For MSX-64 the bound is provable, because the round function is strongly universal, which bounds the maximum expected differential and linear probability after four rounds. For MSX-128,

where a provable bound is not available, the values are loose MILP upper bounds that combine an existing model of modular addition with a bit-level bound on the modular multiplication through its constituent `AND` operations (the 8-round value 2^{-102} quoted above is one of these). These are *distinguisher* bounds, meant to justify the security margin. The designers present them as estimates rather than proofs, computed under the assumption that the carry bits of the multiplication have no effect. No explicit trail is given, so there is nothing from which a key recovery could be built. Their only key recovery type result is the 7-round yoyo attack [BLP15] above, which recovers the secret round functions rather than the master key.

It is still worth asking how far a differential key recovery would reach if their estimates hold and a trail attaining them existed. We peel one round and count subkey candidates in the standard way [BS90, BS91], with the counters treated as Gaussian, for which Selçuk gives the data of a differential attack directly [Sel08, Corollary 2],

$$N = \frac{\left(\sqrt{S_N + 1}\,\Phi^{-1}(P_S) + \Phi^{-1}(1 - 2^{-a})\right)^2}{S_N}\, p^{-1}, \tag{7}$$

where p is the probability of the characteristic, p_r the average probability that a random pair suggests a given key, $S_N = p/p_r$ the signal-to-noise ratio, a the advantage in bits, P_S the success probability and Φ the standard normal distribution function. Here the copy branches of the Type-2 round carry no key, so the differences in W_0 and W_2 are read off the ciphertext and checked against the trail, two 32-bit conditions giving a free filter $\beta = 2^{-64}$. A surviving pair then imposes a single 32-bit equation on the 192-bit subkey, satisfied by about 2^{160} of its 2^{192} values, so a fixed candidate is among them with probability 2^{-32}. Hence $p_r = \beta\, 2^{-32} = 2^{-96}$ and $S_N = 2^{-6}$. The advantage a measures how far the ranking narrows the search, and we need the right key to come first among the 2^{192} candidates, the full advantage $a = 192$, so $\Phi^{-1}(1 - 2^{-192}) \approx 16$. At their eight-round estimate $p = 2^{-102}$ and at $P_S = 1/2$, (7) gives $N = 2^{116}$ pairs, that is $2N = 2^{117}$ chosen plaintexts. That is inside the codebook, so a nine-round key recovery would be possible, and the guessing phase stays far below exhaustive search, so the limit is data rather than time. A ten-round key recovery needs a nine-round trail, and interpolating their eight- and ten-round values, 2^{-102} and 2^{-132}, puts it near 2^{-117}. The same computation then puts the data near 2^{147}, past the 2^{128} codebook. On the designers' own numbers the differential route is therefore data-limited to nine rounds, the same round count our integral attack reaches.

8 Conclusion

In this work, we gave two exact models of modular multiplication modulo 2^n, one for each of the two deterministic properties that automated tools search. The first is for monomial prediction under 3SDP-woU, and it keeps an exact parity for every monomial without any truth table. The second is a deterministic truncated model for impossible differential search, built on the pivot rule of Proposition 1, which we proved is not merely sound but exact for two independent operands. Both are implemented in CLAASP, so a design that uses a modular multiplication can now be analyzed automatically for both properties.

Using them, we gave the first independent cryptanalysis of MSX. For MSX-128, we found a 7-round deterministic integral distinguisher with data complexity 2^{96} and extended it to key recoveries on 8 and 9 of the 18 rounds, which recover the 256-bit master key in about 2^{228}. We also computed upper bounds on algebraic degrees. For impossible differentials, we searched all $16,384$ single-bit pairs at each round count and found $1,984$ at seven rounds and none at eight; thus, at this granularity, our search reaches seven rounds. The designers estimate 11 rounds for this property using a black-box formula and no trail.

In summary, our results provide concrete, explicit constructions in place of bounds and estimates. They only affect a round-reduced MSX, so the security margin of the full cipher

stands. The two views are also complementary. Algebraic resistance does not follow from differential bounds, so a design has to be checked against both.

A natural next step is to extend the same bit-level decomposition to differential and linear characteristics, so that a search sees the actual difference cancellations through the multiplication instead of a black-box round function. It might also be possible that alternative models for modular multiplication exist. Finally, the same approach might also be adapted to other multiplication-based ciphers like IDEA [LM91], MMB [DGV93], MARS [BCD+98], or RC6 [RRSY98].

References

[BBI+15] Subhadeep Banik, Andrey Bogdanov, Takanori Isobe, Kyoji Shibutani, Harunaga Hiwatari, Toru Akishita, and Francesco Regazzoni. Midori: A block cipher for low energy. In *Advances in Cryptology - ASIACRYPT 2015*, volume 9453 of *Lecture Notes in Computer Science*, pages 411–436. Springer, 2015.

[BBS99] Eli Biham, Alex Biryukov, and Adi Shamir. Cryptanalysis of skipjack reduced to 31 rounds using impossible differentials. In *Advances in Cryptology - EUROCRYPT '99*, volume 1592 of *Lecture Notes in Computer Science*, pages 12–23. Springer, 1999.

[BC13] Christina Boura and Anne Canteaut. On the influence of the algebraic degree of f^{-1} on the algebraic degree of G ∘ F. *IEEE Trans. Inf. Theory*, 59(1):691–702, 2013. doi:10.1109/TIT.2012.2214203.

[BCD+98] Carolynn Burwick, Don Coppersmith, Edward D'Avignon, Rosario Gennaro, Shai Halevi, Charanjit Jutla, Stephen M. Matyas Jr., Luke O'Connor, Mohammad Peyravian, David Safford, and Nevenko Zunic. MARS - a candidate cipher for AES. Technical report, IBM Corporation, June 1998. Submission to the AES competition. URL: https://cryptosoft.de/docs/Mars.pdf.

[Ber08] Daniel J. Bernstein. Chacha, a variant of salsa20. Workshop Record of SASC, 2008.

[BGG+23] Emanuele Bellini, David Gérault, Juan Grados, Yun Ju Huang, Rusydi H. Makarim, Mohamed Rachidi, and Sharwan K. Tiwari. CLAASP: A Cryptographic Library for the Automated Analysis of Symmetric Primitives. In Claude Carlet, Kalikinkar Mandal, and Vincent Rijmen, editors, *Selected Areas in Cryptography - SAC 2023 - 30th International Conference, Fredericton, Canada, August 14-18, 2023, Revised Selected Papers*, volume 14201 of *Lecture Notes in Computer Science*, pages 387–408. Springer, 2023. doi:10.1007/978-3-031-53368-6_19.

[BLP15] Alex Biryukov, Gaëtan Leurent, and Léo Perrin. Cryptanalysis of Feistel networks with secret round functions. In *Selected Areas in Cryptography – SAC 2015*, volume 9566 of *Lecture Notes in Computer Science*, pages 102–121. Springer, 2015. doi:10.1007/978-3-319-31301-6_6.

[BRT26] Emanuele Bellini, Mohamed Rachidi, and Sharwan K. Tiwari. CLAASP-MP: An automated MILP framework for monomial prediction. Cryptology ePrint Archive, Paper 2026/735, 2026. URL: https://eprint.iacr.org/2026/735.

[BS90] Eli Biham and Adi Shamir. Differential Cryptanalysis of DES-like Cryptosystems. In Alfred Menezes and Scott A. Vanstone, editors, *Advances*

in Cryptology - CRYPTO '90, 10th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1990, Proceedings, volume 537 of *Lecture Notes in Computer Science*, pages 2–21. Springer, 1990. doi:10.1007/3-540-38424-3_1.

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

[BSS+13] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The SIMON and SPECK Families of Lightweight Block Ciphers. *IACR Cryptol. ePrint Arch.*, page 404, 2013. URL: http://eprint.iacr.org/2013/404.

[DGV93] Joan Daemen, René Govaerts, and Joos Vandewalle. Block ciphers based on modular arithmetic. *Proc. of the 3rd Symposium on the State and Progress of Research in Cryptography*, pages 80–89, 1993.

[DKR97] Joan Daemen, Lars R. Knudsen, and Vincent Rijmen. The block cipher square. In Eli Biham, editor, *Fast Software Encryption, 4th International Workshop, FSE '97, Haifa, Israel, January 20-22, 1997, Proceedings*, volume 1267 of *Lecture Notes in Computer Science*, pages 149–165. Springer, 1997. URL: https://doi.org/10.1007/BFb0052343, doi:10.1007/BFB0052343.

[DL22] Patrick Derbez and Baptiste Lambin. Fast MILP models for division property. *IACR Trans. Symmetric Cryptol.*, 2022(2):289–321, 2022. URL: https://doi.org/10.46586/tosc.v2022.i2.289-321, doi:10.46586/TOSC.V2022.I2.289-321.

[DS09] Itai Dinur and Adi Shamir. Cube attacks on tweakable black box polynomials. In Antoine Joux, editor, *Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings*, volume 5479 of *Lecture Notes in Computer Science*, pages 278–299. Springer, 2009. doi:10.1007/978-3-642-01001-9_16.

[HHLW24] Jiahui He, Kai Hu, Hao Lei, and Meiqin Wang. Massive superpoly recovery with a meet-in-the-middle framework - improved cube attacks on trivium and kreyvium. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology - EUROCRYPT 2024 - 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26-30, 2024, Proceedings, Part I*, volume 14651 of *Lecture Notes in Computer Science*, pages 368–397. Springer, 2024. doi:10.1007/978-3-031-58716-0_13.

[HHPW22] Jiahui He, Kai Hu, Bart Preneel, and Meiqin Wang. Stretching cube attacks: Improved methods to recover massive superpolies. In Shweta Agrawal and Dongdai Lin, editors, *Advances in Cryptology - ASIACRYPT 2022 - 28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 5-9, 2022, Proceedings, Part IV*, volume 13794 of *Lecture Notes in Computer Science*, pages 537–566. Springer, 2022. doi:10.1007/978-3-031-22972-5_19.

[HLM+20] Yonglin Hao, Gregor Leander, Willi Meier, Yosuke Todo, and Qingju Wang. Modeling for three-subset division property without unknown subset - improved cube attacks against trivium and grain-128aead. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020 - 39th*

Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part I, volume 12105 of *Lecture Notes in Computer Science*, pages 466–495. Springer, 2020. doi:10.1007/978-3-030-45721-1_17.

[HST+21] Kai Hu, Siwei Sun, Yosuke Todo, Meiqin Wang, and Qingju Wang. Massive superpoly recovery with nested monomial predictions. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6-10, 2021, Proceedings, Part I*, volume 13090 of *Lecture Notes in Computer Science*, pages 392–421. Springer, 2021. doi:10.1007/978-3-030-92062-3_14.

[HSWW20] Kai Hu, Siwei Sun, Meiqin Wang, and Qingju Wang. An algebraic formulation of the division property: Revisiting degree evaluations, cube attacks, and key-independent sums. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part I*, volume 12491 of *Lecture Notes in Computer Science*, pages 446–476. Springer, 2020. doi:10.1007/978-3-030-64837-4_15.

[HW19] Kai Hu and Meiqin Wang. Automatic search for a variant of division property using three subsets. In Mitsuru Matsui, editor, *Topics in Cryptology - CT-RSA 2019 - The Cryptographers' Track at the RSA Conference 2019, San Francisco, CA, USA, March 4-8, 2019, Proceedings*, volume 11405 of *Lecture Notes in Computer Science*, pages 412–432. Springer, 2019. doi:10.1007/978-3-030-12612-4_21.

[HY24] Kai Hu and Trevor Yap. Perfect monomial prediction for modular addition. *IACR Trans. Symmetric Cryptol.*, 2024(3):177–199, 2024. URL: https://doi.org/10.46586/tosc.v2024.i3.177-199, doi:10.46586/TOSC.V2024.I3.177-199.

[Knu94] Lars R. Knudsen. Truncated and higher order differentials. In Bart Preneel, editor, *Fast Software Encryption: Second International Workshop. Leuven, Belgium, 14-16 December 1994, Proceedings*, volume 1008 of *Lecture Notes in Computer Science*, pages 196–211. Springer, 1994. doi:10.1007/3-540-60590-8_16.

[Knu98] Lars R. Knudsen. DEAL - A 128-bit Block Cipher. Technical Report 151, Department of Informatics, University of Bergen, Norway, February 1998.

[LM91] Xuejia Lai and James L. Massey. A proposal for a new block encryption standard. In Ivan Bjerre Damgård, editor, *Advances in Cryptology — EUROCRYPT '90*, pages 389–404, Berlin, Heidelberg, 1991. Springer Berlin Heidelberg.

[MSR+26] Kazuhiko Minematsu, Tomoyasu Suzaki, Mostafizar Rahman, Sahiba Suryawanshi, and Takanori Isobe. MSX: Lightweight block ciphers for microcontrollers with high-assurance against differential and linear attacks. *IACR Communications in Cryptology*, 2(4), 2026. doi:10.62056/av11c3c2h.

[RRSY98] Ronald L Rivest, Matthew JB Robshaw, Ray Sidney, and Yiqun Lisa Yin. The rc6tm block cipher. In *First advanced encryption standard (AES) conference*, page 16, 1998.

[Sel08] Ali Aydin Selçuk. On probability of success in linear and differential cryptanalysis. *Journal of Cryptology*, 21(1):131–147, 2008. doi:10.1007/s00145-007-9013-7.

[TIHM18] Yosuke Todo, Takanori Isobe, Yonglin Hao, and Willi Meier. Cube attacks on non-blackbox polynomials based on division property. *IEEE Trans. Computers*, 67(12):1720–1736, 2018. doi:10.1109/TC.2018.2835480.

[TM16] Yosuke Todo and Masakatu Morii. Bit-Based Division Property and Application to Simon Family. In Thomas Peyrin, editor, *Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*, volume 9783 of *Lecture Notes in Computer Science*, pages 357–377. Springer, 2016. doi:10.1007/978-3-662-52993-5_18.

[Tod15] Yosuke Todo. Structural Evaluation by Generalized Integral Property. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 287–314. Springer, 2015. doi:10.1007/978-3-662-46800-5_12.

[Vau03] Serge Vaudenay. Decorrelation: A theory for block cipher security. *Journal of Cryptology*, 16(4):249–286, 2003.

[WHG+19] Senpeng Wang, Bin Hu, Jie Guan, Kai Zhang, and Tairong Shi. Milp-aided method of searching division property using three subsets and applications. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part III*, volume 11923 of *Lecture Notes in Computer Science*, pages 398–427. Springer, 2019. doi:10.1007/978-3-030-34618-8_14.

[XYL18] Yiran Xing, Hailun Yan, and Xuejia Lai. New observation on division property: Simplifying models of basic operations and modeling modular multiplication operation. In Ali Emrouznejad and Zhihong Qian, editors, *The 2nd International Conference on Computer Science and Application Engineering, CSAE 2018, Hohhot, China, October 22-24, 2018*, pages 8:1–8:6. ACM, 2018. doi:10.1145/3207677.3278003.

[XZBL16] Zejun Xiang, Wentao Zhang, Zhenzhen Bao, and Dongdai Lin. Applying MILP method to searching integral distinguishers based on division property for 6 lightweight block ciphers. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 648–678, 2016. doi:10.1007/978-3-662-53887-6_24.