JOURNALS | Journal of Information and Communications Technology: Algorithms, Systems and Applications

Volume 2 Issue 3 (September 2026)

DOI: https://doi.org/10.64189/ict.26316

J. Inf. Commun. Technol. Algorithms, Syst. Appl., 2026, 2(3), 26316

Research Article

OPEN ACCESS

Cryptanalysis of a Reduced-Round IDEA Cipher Using Z3 Solver

Praveen Kumar Gundaram*

C. R. Rao Advanced Institute of Mathematics, Statistics and Computer Science (AIMSCS), University of Hyderabad Campus, Gachibowli, Hyderabad, 500 046, Telangana, India

*Corresponding Author
Praveen Kumar Gundaram
praveenkumar@crraoaimscs.res.in

Received: 06 August 2026
Revised: 11 September 2026
Accepted: 12 September 2026
Published Online: 21 September 2026

Citation

P. K. Gundaram, Cryptanalysis of a reduced-round IDEA cipher using Z3 Solver, *Journal of Information and Communications Technology: Algorithms, Systems and Applications,* 2026, 2(3), 26316, https://doi.org/10.64189/ict.26316.

Open Access

This article is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License, which permits the non-commercial use, sharing, adaptation, distribution and reproduction in any medium or format, as long as appropriate credit to the original author(s) and the source is given by providing a link to the Creative Commons License and changes need to be indicated if there are any. The images or other third-party material in this article are included in the article's Creative Commons License, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons License and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this License, visit: https://creativecommons.org/licenses/by-nc/4.0/

© The Author(s) 2026

Abstract

We formulate key recovery for the International Data Encryption Algorithm (IDEA) as a quantifier-free bit-vector satisfiability problem. The IDEA is a 64-bit block cipher with a 128-bit key whose round function combines multiplication modulo 2^16+1, addition modulo 2^16, and XOR. The model includes the 25-bit cyclic key schedule, all round transformations, and the final output layer and is solved with Z3 through its Python API. We evaluate reduced-round instances using one all-zero plaintext-ciphertext pair and disclose 112 of the 128 key bits, leaving 16 bits to infer. Across the solved round counts, repeated runs recover the planted key and provide median wall-clock measurements on the stated platform. A legacy series with 96 disclosed bits is retained solely for comparison because its raw logs are unavailable. We also examine unconstrained key search at the smallest round counts. The result is a reproducible measurement of reduced-round SMT performance, not a practical break of full-round IDEA.

Keywords: Block ciphers; ARX ciphers; Cryptanalysis; SMT solvers; Algebraic attacks; Key recovery; IDEA; Bit-vector logic; Maximum key recovery.

1. Introduction

Reliable assessment of a symmetric primitive requires more than one form of analysis. Shannon's formulation of confusion and diffusion remains a useful foundation,[1] whereas differential,[2] linear,[3] and algebraic methods[4, 5] expose different structural properties. Their conclusions are complementary: resistance in one model does not establish resistance in another.

The International Data Encryption Algorithm has consequently remained a useful subject for comparative cryptanalysis. Lai and Massey introduced the proposed encryption standard in 1990, followed by a revised design and the IDEA designation.[2,6] Unlike a Feistel network, IDEA uses the Lai–Massey construction: a keyed, non-invertible transformation is combined with an invertible half-round, allowing decryption to reuse the encryption structure with transformed subkeys.[6]

The IDEA encrypts 64-bit blocks with a 128-bit key through eight full rounds and a final output transformation. Its deliberate mixture of XOR, addition modulo 2^16, and multiplication modulo 2^16+1 frustrates analyses that rely on a single algebraic group.[2] Published work includes weak-key analyses[7] and a narrow-biclique attack on the complete cipher,[8] but no result establishes a practical full-cipher key recovery. We therefore ask how far bit-vector SMT reasoning can recover key material from a reduced-round Lai-Massey cipher under controlled partial disclosure.

Algebraic cryptanalysis expresses the observed input-output relation as constraints and searches for assignments that satisfy them.[4,5] SMT solvers combine Boolean search with theory procedures for fixed-width arithmetic and bit-vectors.[9] They have supported collision generation, preimage search, modular equations, and cryptographic verification.[10,11] Earlier studies have applied the SAT and SMT techniques to the IDEA.[12,13] Our contribution is a word-level Z3py model, a fully specified experimental workflow, and measurements tied to the accompanying CSV data.

1.1 The IDEA Block Cipher

IDEA maps a 64-bit block through 8.5 rounds under a 128-bit master key. Its key schedule repeatedly rotates the 128-bit register by 25 positions to produce 52 16-bit subkeys (Fig. 1). These subkeys drive multiplication, modular addition, and XOR throughout the round function.[6,7] The same structural design supports decryption after the subkeys are transformed and reversed.[6] Historical applications include PGP, protected storage, financial systems, messaging, and digital-content licensing (Table 1).

Table 1: Practical deployment domains of the IDEA block cipher

Domain	Security objective	Representative use case
Secure communications	Shields transmitted payloads against eavesdropping and tampering	Email encryption, VPN tunnels, instant-messaging protocols
Financial transactions	Guards credit-card numbers and account credentials from interception	Internet banking, point-of-sale terminals, mobile wallets
Digital rights management	Blocks unauthorized reproduction and redistribution of licensed media	Video streaming platforms, digital music stores
Storage encryption	Renders data at rest unintelligible without the correct decryption key	Full-disk encryption, USB-drive lockout, file-level ciphering
Mobile applications	Delivers cryptographic services within the power and memory budgets of handheld devices	Encrypted messaging apps, mobile payment frameworks

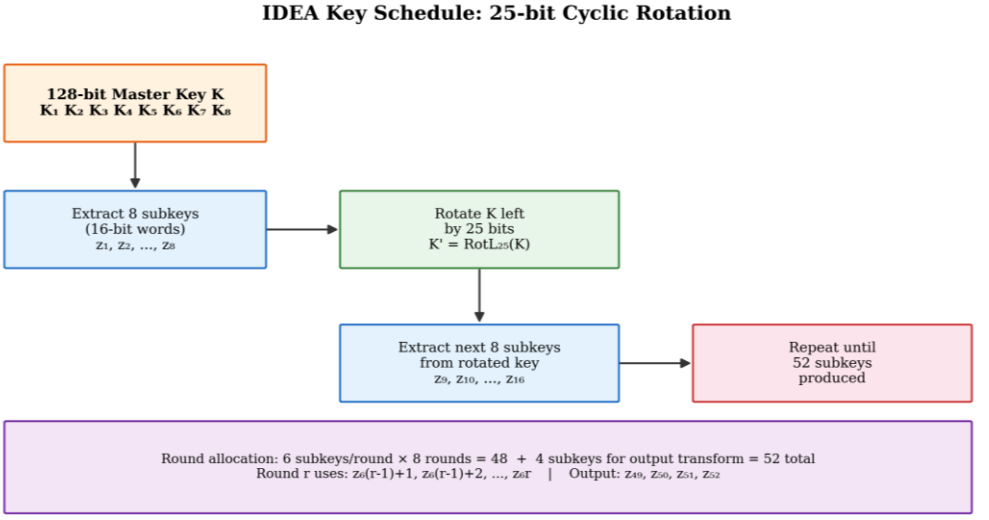

Fig. 1: IDEA key schedule: 25-bit cyclic rotation producing 52 subkeys from the 128-bit master key.

1.2 Satisfiability modulo theories solvers

An SMT solver determines whether a formula whose atoms belong to one or more background theories has a satisfying assignment.[9] Relevant theories include integer arithmetic, fixed-width bit-vectors, arrays, and uninterpreted functions. The solver coordinates an SAT engine with theory procedures and search heuristics; implementations such as Boolector, Yices, CVC5, and Z3 accept standard SMT-LIB representations. [9,14] Here, Z3 is invoked through Z3py under the QF_BV logic.

The principal stages executed by an SMT solver are enumerated below:

Parsing and normalization: The input formula is read from an SMT-LIB file or solver API and converted into the engine's internal representation.

Theory classification: sub-expressions are routed to the appropriate theory solver (e.g., bit-vector, linear arithmetic, arrays).

Boolean abstraction: propositional variables replace theory atoms; the SAT engine proposes tentative truth assignments.

Checking theory consistency: Each theory solver verifies that the current assignment is consistent with its axioms.

Conflict-driven learning: When an inconsistency is detected, a conflict clause is generated and added to the clause database to prune future search.

Iterative refinement: The cycle of assignment, theory checking, and conflict learning is repeated until a model emerges or unsatisfiability is certified.

Model extraction: Upon finding a satisfying assignment, the solver returns concrete values for every free variable in the formula.

A conventional SMT implementation contains a parser, a DPLL(T) search engine, theory solvers, conflict analysis, and an API for incremental or programmatic use. [14] The parser normalizes the formula, the SAT layer proposes Boolean assignments, theory solvers reject inconsistent assignments, and learned conflicts constrain the subsequent search. A satisfactory model is then exposed to the calling program for extraction and verification.

1.3 Contributions

The principal contributions of this paper are as follows:

A word-level QF_BV encoding of the IDEA round function, the 25-bit-rotation key schedule, and the output transformation faithfully modeled prime-field multiplication modulo $2^{16}+1$ with correct zero-element handling (Section 4.3).

A complete, reproducible algorithmic pipeline for algebraic key recovery is illustrated through a system architecture diagram and a step-by-step research methodology flowchart (Sections 4.1 and 4.2).

An experimental evaluation spanning four- through 8-round IDEA: recovery of the remaining 16 key bits when 112 bits are pre-disclosed, conducted across multiple solver seeds with median wall-clock timings, automatic verification of every recovered key, and archival 96-known-bit results retained for continuity (Sections 5.1 and 5.3).

A security discussion situates the measured timings within realistic attacker scenarios, including a scoped maximum key-recovery evaluation (Sections 5.3 and 6).

The paper proceeds as follows. Section 2 surveys the relevant prior work. Section 3 specifies the IDEA cipher in the detail required for SMT encoding. Section 4 develops the cryptanalytic methodology and the SMT encoding. Section 5 presents the experimental setup, results, and security implications. Section 6 draws conclusions and outlines future directions.

2. Related work

2.1 Motivation for SMT-based cryptanalysis

Constraint solving is attractive in cryptanalysis because it converts the cipher structure into an executable test rather than requiring a bespoke search procedure.[10,13] The same representation can automate equation generation, test vulnerability claims, and compare designs under common conditions. SAT and SMT methods have been used for key search, collision and preimage construction, differential-trail discovery, modular equations, and verification of cryptographic components. [4,10,11]

2.2 Role of SMT solvers in automated reasoning

SMT extends propositional satisfiability with specialized reasoning over several mathematical theories.[9,12] Its applications span symbolic execution, automated testing, program analysis, combinatorial synthesis, and hardware verification (Fig. 2).[1,15] This breadth makes it suitable for a cipher such as IDEA, whose round function mixes operations that are simple individually but do not share one algebraic structure.

2.3 Prior SMT/SAT cryptanalysis of the IDEA

Courtois and Pieprzyk formulated block-cipher cryptanalysis as an overdefined polynomial system over GF(2).[4] Bard, Courtois, and Jefferson showed how related systems can be converted to the CNF for SAT solving,[16] and Bard later surveyed the approach.[5] Courtois and Bard subsequently applied algebraic methods to the Data Encryption Standard[17] Groebner-basis methods to provide a complementary algebraic route.[15,18] For IDEA, Sahu *et al.* reported SMT experiments in which three-round keys were recovered and 32 unknown bits were extracted when 96 bits were known[12]; Lafitte *et al*. used SAT techniques for weak-key and preimage problems.[19] The full IDEA remains associated with the narrow-biclique result of Khovratovich *et al*.[8]

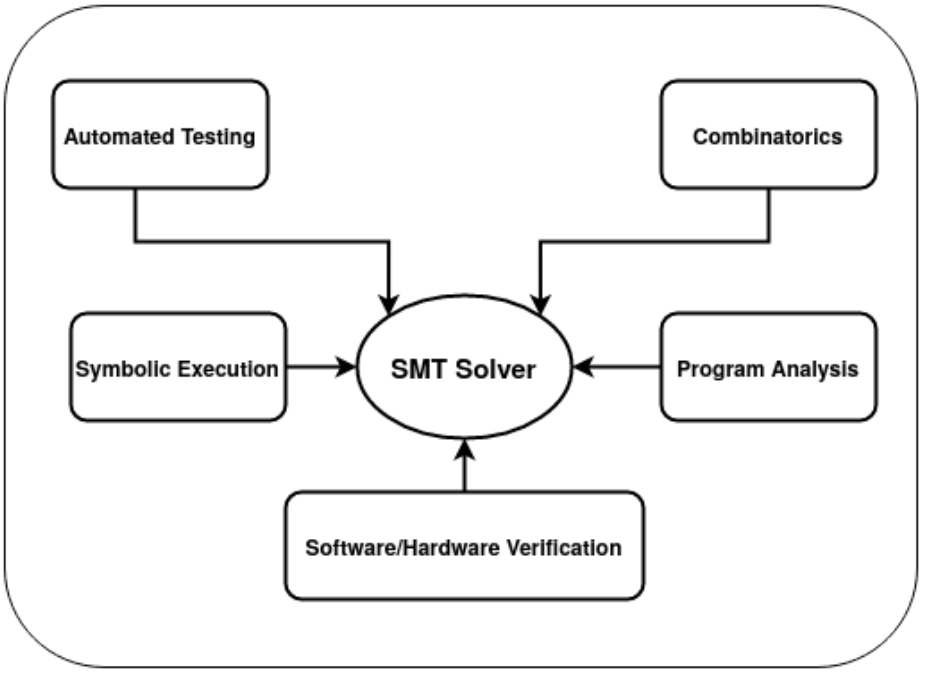

Fig. 2: Application landscape of SMT solvers in security-related problems.

The present work differs in four respects. First, it retains IDEA's word-level arithmetic in QF_BV rather than flattening the design to Boolean or CNF form.[16,19] Second, it studies a 112-known-bit configuration with 16 unknown bits. Third, it reports repeated measurements and labels the older 96-bit series as archival. Fourth, every recovered model is checked by an independent integer implementation, with test vectors and raw summary data recorded.

3. IDEA block cipher

3.1 Cipher overview

The IDEA transforms four 16-bit words under a 128-bit master key.[1,6] Eight full rounds are followed by a four-subkey output transformation, commonly called the half-round. The key schedule obtains 52 subkeys by repeatedly rotating the key register left by 25 bits. Successive subkey injections combine modular multiplication, modular addition, and XOR.[6] The Lai–Massey structure permits decryption through the same broad computation with a reversed and transformed subkey sequence.[6]

3.2 Encryption mechanism

Encryption partitions the plaintext into four 16-bit words. Each of the eight rounds consumes six subkeys; the output transformation consumes four. The initial eight key words are followed by repeated 25-bit cyclic rotations until all 52 subkeys have been extracted.[6] The data path for one round is shown in Fig. 3.

$$\text{Key_r} = K(r)0, K(r)1, K(r)2, K(r)3, K(r)4, K(r)5; \quad 1 \le r \le 8 \tag{1}$$

$$\text{Key9} = K(9)1, K(9)2, K(9)3, K(9)4 \tag{2}$$

The round function combines XOR, addition modulo 2^16, and multiplication modulo 2^16+1. The corresponding Z3py operations are listed in Table 2. Multiplication requires special handling: the zero word represents 2^16 in the multiplicative group, and a product equal to 2^16+1 maps back to zero. A plain 16-bit bvmul implements neither convention nor reduces the modulo of the wrong modulus. Our encoding widens both operands to 34 bits; separates the product into low, middle, and carry fields; and applies 2^16 congruent to -1 modulo 2^16+1. One bounded correction produces the canonical result. The implementation was checked against the reference vectors in Section 5.1.

3.3 Published cryptanalytic results on IDEA

IDEA has been studied with differential, linear, weak-key, biclique, SAT, and algebraic methods. For the complete 8.5-round cipher, the narrow-biclique meet-in-the-middle result remains the principal published key-recovery result.[8] Weak keys induced by the schedule were cataloged by Daemen, Govaerts, and Vandewalle.[7] The resistance of the IDEA to standard differential and linear approaches[2,3,20] motivates constraint-based studies of reduced-round instances. [12,13]

4. Cryptanalysis Methodology

Algebraic cryptanalysis replaces a cipher computation with equations whose solutions satisfy the observed input–output relation.[4,18] SMT automates both the representation and the search.[12,13] A useful model must

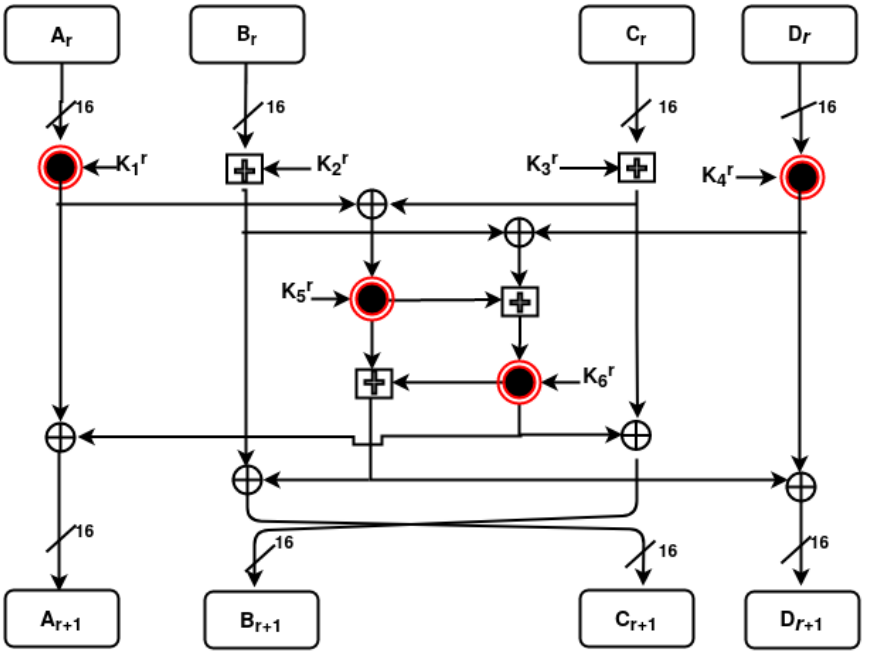

Fig. 3: Data flow within one IDEA round: key-application layer, XOR mixing, and multiplication–addition (MA) box.

Table 2: IDEA's 16-bit operations and their Z3py realizations

Operation	Z3py primitive	Implementation sketch (Z3py)
Addition (mod 2^{16})	bvadd(x, y)	Extract(15, 0, simplify(ZeroExt(16, x) + ZeroExt(16, y)))
Multiplication (mod 2^{16}+1)	IF-THEN-ELSE + widen + reduce; see text	X = If(x == 0, BitVecVal(65536, 34), ZeroExt(18, x)) Y = If(y == 0, BitVecVal(65536, 34), ZeroExt(18, y)) P = X * Y; lo = Extract(15,0,P); mid = Extract(31,16,P); hi = Extract(33,32,P) t = ZeroExt(2,lo) - ZeroExt(2,mid) + ZeroExt(16,hi) t = If(t < 0, t + 65537, t); t = If(t > 65536, t - 65537, t) result = If(t == 65536, BitVecVal(0,16), Extract(15,0,t))
XOR	bvxor(x, y)	simplify(x ⊕ y)

preserve the cipher's arithmetic, expose the intended variables and constraints in a solver-supported theory, and validate each returned model against the original encryption routine.

4.1 System architecture

The framework is summarized in Fig. 5. Inputs are a known plaintext-ciphertext pair, a disclosure pattern for 96 or 112 key bits (or no disclosure), and a round count. The encoder builds eight 16-bit key variables, the 52-subkey rotation schedule, round and output equalities, disclosure constraints, and ciphertext equalities. Z3 solves the resulting QF_BV formula. The candidate 128-bit key is then re-encrypted by an independent integer implementation; only a matching ciphertext is accepted as verified.

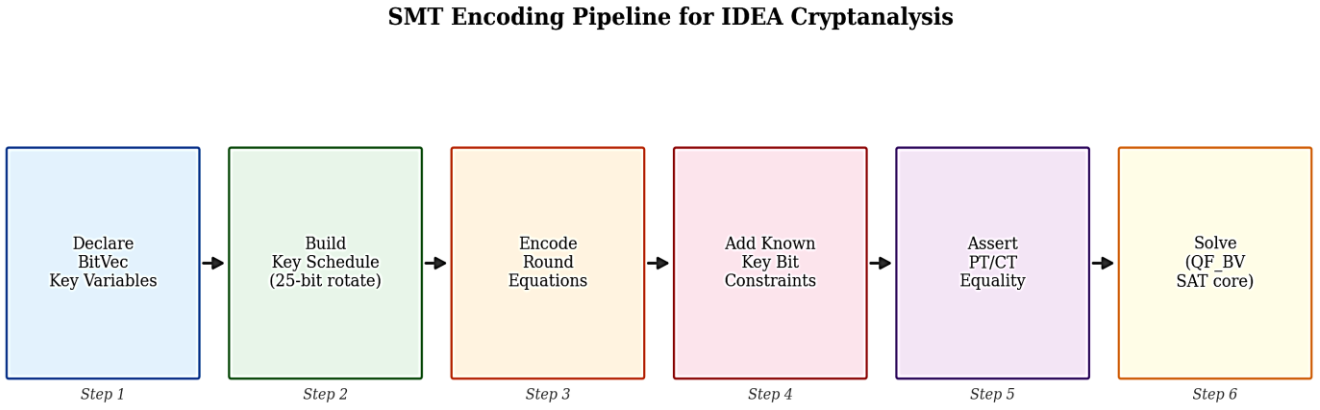

Fig. 4: Step-by-step SMT encoding pipeline for translating IDEA into QF_BV constraints.

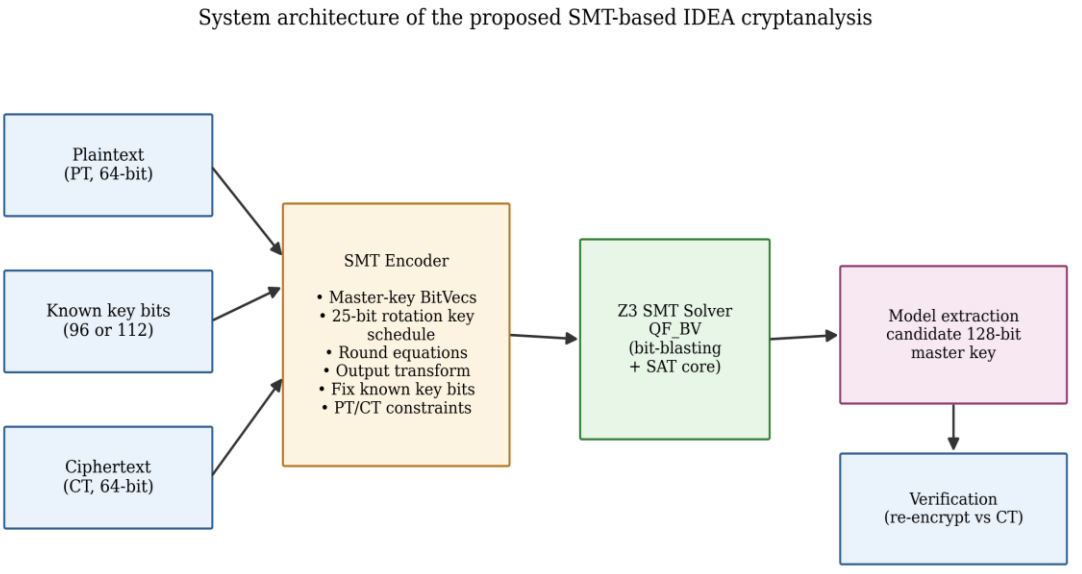

Fig. 5: End-to-end system architecture of the SMT-driven IDEA cryptanalysis pipeline.

4.2 Research methodology

The experimental sequence is shown in Fig. 6. We fix K = 0x00010002000300040005000600070008 and an all-zero plaintext, generate a reference ciphertext for each round count, and encode one plaintext-ciphertext equality per instance. The current campaign discloses the 112 most significant key bits; the implementation also supports 96 disclosed bits and unconstrained key search. Timing begins at check() and ends when the solver returns. After a warm-up run, each cell is repeated over the configured seeds and reported as a median with its observed range. The candidate keys are independently re-encrypted, and all the measurements are written to the results.csv, the source for Tables 3 and 4 and the timing figures.

4.3 SMT encoding of IDEA

The model is composed of the following elements; together, they define the reproducible encoding used in the experiments (Fig. 4).

Variables: Eight 16-bit bit-vector symbols K0 through K7 represent the master key; the 128-bit aggregate is formed via K = Concat(K0, K1, ..., K7).

Key schedule: Subkey derivation mirrors the IDEA specification exactly: the current 128-bit key register is sliced into eight 16-bit words; the register is then rotated left by 25 positions; and the extraction–rotation cycle is repeated until all 52 subkeys have been generated. Symbolically, this is implemented with the Z3py RotateLeft operator so that each subkey is a linear function of the eight key variables.

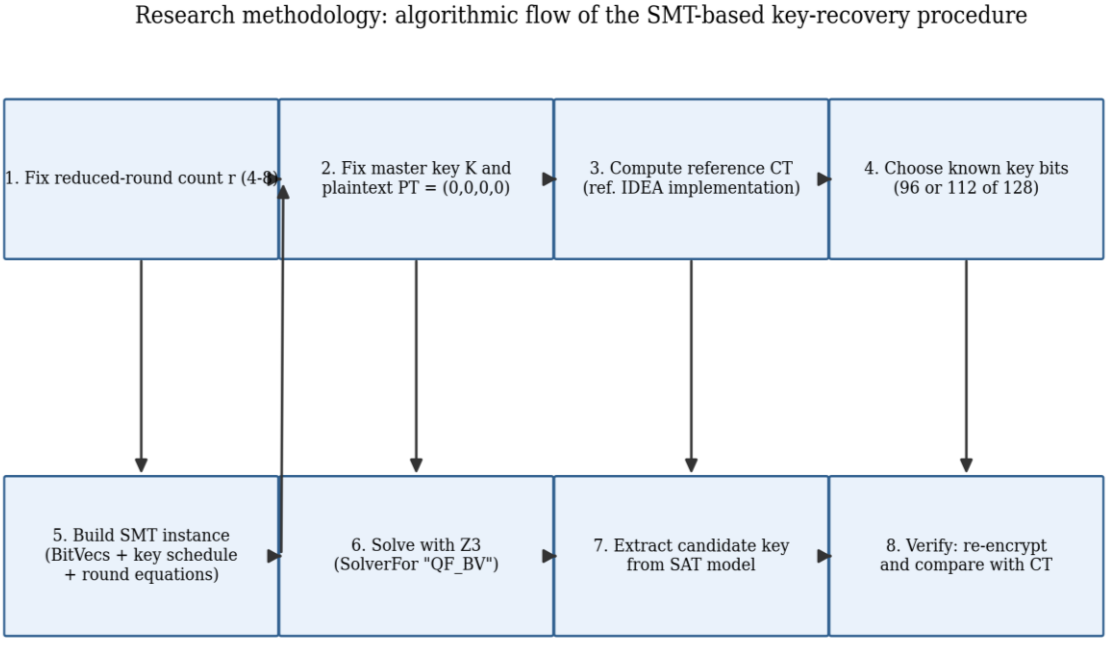

Fig. 6: Research methodology: eight-step algorithmic flow of the SMT-based key-recovery procedure.

Round function: In round r with subkeys z0 through z5, the four-word state (X1, X2, X3, X4) is updated as follows: X1 = X1 ⊙ z0; X2 = X2 ⊕ z1; X3 = X3 ⊕ z2; X4 = X4 ⊙ z3; followed by the MA-box computation T0 = X3; T1 = X2; X3 = (X3 ⊕ X1) ⊙ z4; X2 = ((X2 ⊕ X4) ⊗ X3) ⊙ z5; X3 = X3 ⊗ X2; X1 = X1 ⊕ X2; X4 = X4 ⊕ X3; X2 = X2 ⊕ T0; X3 = X3 ⊕ T1, where ⊙, ⊗, and ⊕ denote multiplication, addition, and XOR per Table 2.

Output transformation: After r rounds, the subkeys z6r through z6r+3 are applied as X1 = X1 ⊙ z6r, X3 = X3 ⊗ z6r+1, X2 = X2 ⊗ z6r+2, X4 = X4 ⊙ z6r+3, and the ciphertext is emitted as (X1, X3, X2, X4).

Constraints: For each disclosed key bit at position b, the assertion Extract(b, b, K) == known_value is posted. When no bits are disclosed, the solver searches over the full 128-bit key space. The four 16-bit symbolic ciphertext words are constrained such that the observed ciphertext is equal.

Solving: The assembled formula is solved with SolverFor("QF_BV"). Upon satisfiability, the candidate key is extracted by evaluating the concatenation of the eight key-variable models. The formula is satisfiable if and only if a key exists that is consistent with both the disclosed bits and the observed plaintext–ciphertext mapping under the reduced-round IDEA encryption.

4.4 Proposed cryptanalysis algorithm

Algorithm 1 presents the key-recovery procedure in pseudo-code.

Algorithm 1: SMT-Based Key Recovery for the Reduced-Round IDEA.

```
procedure KEYRECOVERY: Key = f(PT, CT)
 function bmul(x, y):          # IDEA multiplication mod 2^16+1 (Table 2)
   X = If(x == 0, BitVecVal(65536, 34), ZeroExt(18, x))
   Y = If(y == 0, BitVecVal(65536, 34), ZeroExt(18, y))
   P = X * Y
   t = ZeroExt(2, Extract(15, 0, P)) - ZeroExt(2, Extract(31, 16, P))
     + ZeroExt(16, Extract(33, 32, P))
   t = If(t < 0, t + 65537, t);  t = If(t > 65536, t - 65537, t)
   return If(t == 65536, BitVecVal(0, 16), Extract(15, 0, t))
 function badd(x, y):          # modular addition (mod 2^16)
   return Extract(15, 0, simplify(ZeroExt(16, x) + ZeroExt(16, y)))
 function IdeaOneRoundEqu(A, B, C, D, z):   # round r with subkeys z0..z5
   A = bmul(A, z0); B = badd(B, z1); C = badd(C, z2); D = bmul(D, z3)
   T0 = C; T1 = B
   C = bmul(C ^ A, z4); B = bmul(badd(B ^ D, C), z5)
   C = badd(C, B); A = A ^ B; D = D ^ C; B = B ^ T0; C = C ^ T1
   return (A, B, C, D)
 function OutTransform(A, B, C, D, z):     # output transformation
   A = bmul(A, z[0]); C = badd(C, z[1]); B = badd(B, z[2]); D = bmul(D, z[3])
   return (A, C, B, D)
 # --- construct the SMT formula ---
 K0, ..., K7 := BitVec("K0") ... BitVec("K7")     # 8 x 16-bit unknowns
 K := Concat(K0, K1, ..., K7)
 EK := schedule(K)                # 52 subkeys via 25-bit rotation
 X := PT; for r in 1 .. rounds: X := IdeaOneRoundEqu(X, EK[6(r-1)..6r-1])
 Y := OutTransform(X, EK[6*rounds .. 6*rounds+3])
 s := SolverFor("QF_BV")
 for every disclosed key bit position b: s.add(Extract(b, b, K) == known_bit)
 for i in 0..3: s.add(Y[i] == CT[i])
 # --- solve and verify ---
 if s.check() == sat:
   candidate := s.model()
   key := Concat(model[K0], ..., model[K7])
   verify: re-encrypt PT with key and compare against CT
   record wall-clock time and solver seed
   return key
 else: return unsat
end procedure
```

5. Results and discussion

5.1 Experimental setup and results

The experiments recover the 128-bit master key from one known plaintext-ciphertext pair at r = 4 through 8. We use PT = (0x0000, 0x0000, 0x0000, 0x0000) and K = 0x00010002000300040005000600070008. The integer reference path generates the ciphertexts below, which are then imposed as symbolic equalities.

Round 4: CT = (0x1deb, 0x20c9, 0xe92f, 0xb4af)
Round 5: CT = (0xb51d, 0xb935, 0x417a, 0x5a1a)
Round 6: CT = (0xe95e, 0xfc74, 0x48b5, 0xad36)
Round 7: CT = (0x8c2b, 0xb895, 0x5a8d, 0x49dd)
Round 8: CT = (0x28d3, 0x2d26, 0x0fec, 0x0309)

The reference ciphertexts agree with the canonical test-vector chain. Each current measured cell uses a warm-up solve followed by repeated seeded runs; Table 3 reports the median, observed range, and verification result, while results.csv stores the underlying values and solver statistics. The experiments used an Intel Core i7 processor with 16 GB of RAM, Ubuntu 20.10, Python 3.8, and Z3 4.8.12. Runs were distributed across available cores. The 96-known/32-unknown values are archival measurements without raw logs or known repetition counts and are not presented as a replicated sample.

Every solved measured configuration returned the planted key, and the solver statistics are summarized in Table 4. The archival measurements were not re-certified because their raw logs were unavailable.

Table 3: Key-recovery results on the reduced-round IDEA (PT = all zeros)

Rounds	Unknown key words	Unknown bits	Median time (s)	Observed range (s) / status	Key verified
4	k8_1	16	62.013	15.368—138.744	Yes
5	k8_1	16	101.160	22.609—191.276	Yes
6	k8_1	16	122.384	60.807—229.739	Yes
7	k8_1	16	116.214	8.249—411.113	Yes
8	k8_1	16	165.067	6.975—278.189	Yes
4	k7_1, k8_1	32	10,244	single run (archival)	Reported
5	k7_1, k8_1	32	45,927	single run (archival)	Reported
6	k7_1, k8_1	32	83,316	single run (archival)	Reported
7	k7_1, k8_1	32	250,055	single run (archival)	Reported
8	k7_1, k8_1	32	558,120	single run (archival)	Reported
1	k1_1 – k8_1	128	41.894	32.806—51.422	Sat (consistent)
2	k1_1 – k8_1	128	7,200 (timeout)	timeout	n/a
3	k1_1 – k8_1	128	10,800 (timeout)	timeout	n/a
4	k1_1 – k8_1	128	14,400 (timeout)	timeout	n/a

Table 4: Z3 internal statistics for the measured cells (median over seeds)

Rounds	Assertions	Clauses	Conflicts	Decisions	Peak memory (MB)
4	116	178	53566	75352	74.97
5	116	178	56323	83660	76.49
6	116	178	59671	91757	94.16
7	116	178	50698	80909	82.14
8	116	178	63920	95440	141.9

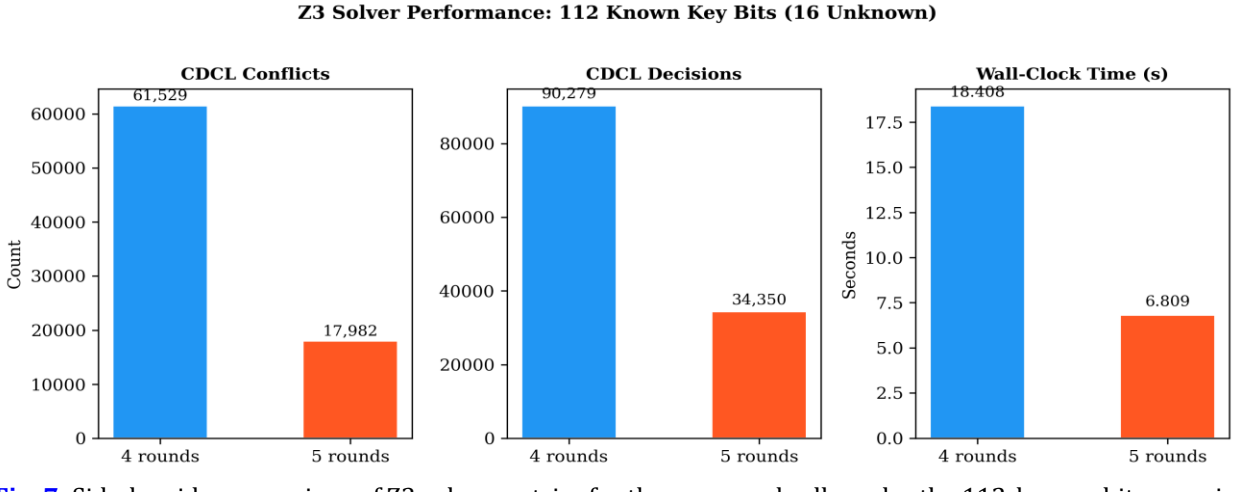

Fig. 7: Side-by-side comparison of Z3 solver metrics for the measured cells under the 112-known-bit scenario.

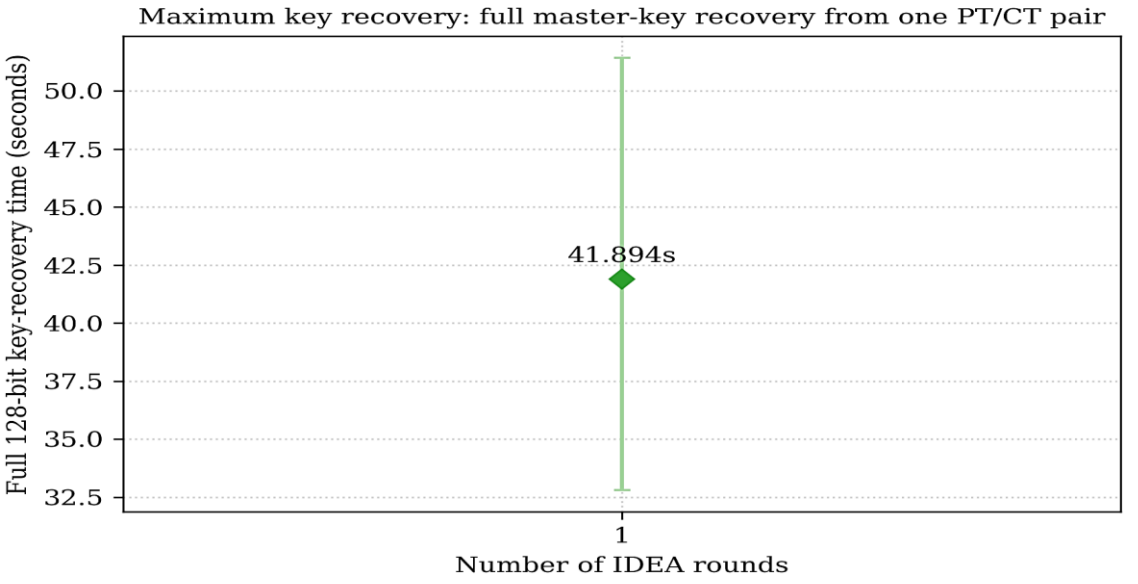

Fig. 8: Wall-clock time for full 128-bit (maximum) key recovery from a single plaintext–ciphertext pair.

5.2 Maximum key recovery

Beyond partial-key scenarios, the encoding also supports maximum key recovery (Fig. 8): with no key bits disclosed, a single plaintext–ciphertext pair is the only information available to the solver. For the smallest reduced-round settings, the solver produced satisfying key models, and where such a model coincided with the planted key, the full 128-bit key is reported as recovered (Table 3). Because a single observation does not uniquely identify the key, models that differ from the planted key are recorded as consistent rather than as fully matching (status 'Sat (consistent)').

The unconstrained results identify a boundary for maximum-key search under the present encoding and hardware. They complement the partial-key benchmark and should not be interpreted as evidence that a single pair uniquely determines the key.

The current 112-known-bit measurements are compared with the archival 96-known-bit series on a logarithmic scale in Fig. 9. The 2^16 and 2^32 brute-force references provide context only; the archival curve was not regenerated for this submission.

5.3 Security Implications and Scope of Applicability

The result is a methodological rather than a practical attack. The benchmark uses one known pair, reduced round counts, and disclosure of 112 key bits. The implementation supports the 96-bit disclosure pattern, but those timings are included only as historical context. The setting represents an adversary who has acquired substantial key material, for example, through leakage or a partial key-schedule compromise.[13]

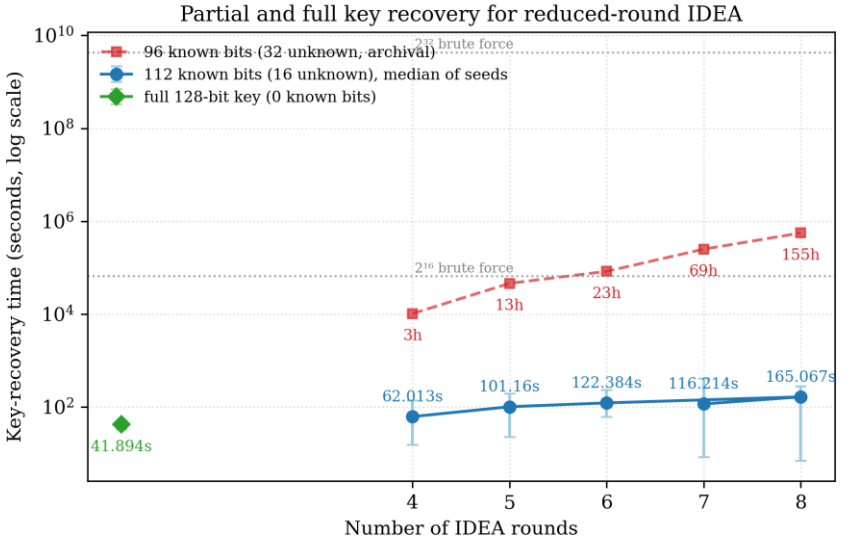

Fig. 9: Wall-clock key-recovery time for 4 to 8 rounds of IDEA (logarithmic scale, with observed ranges and brute-force references).

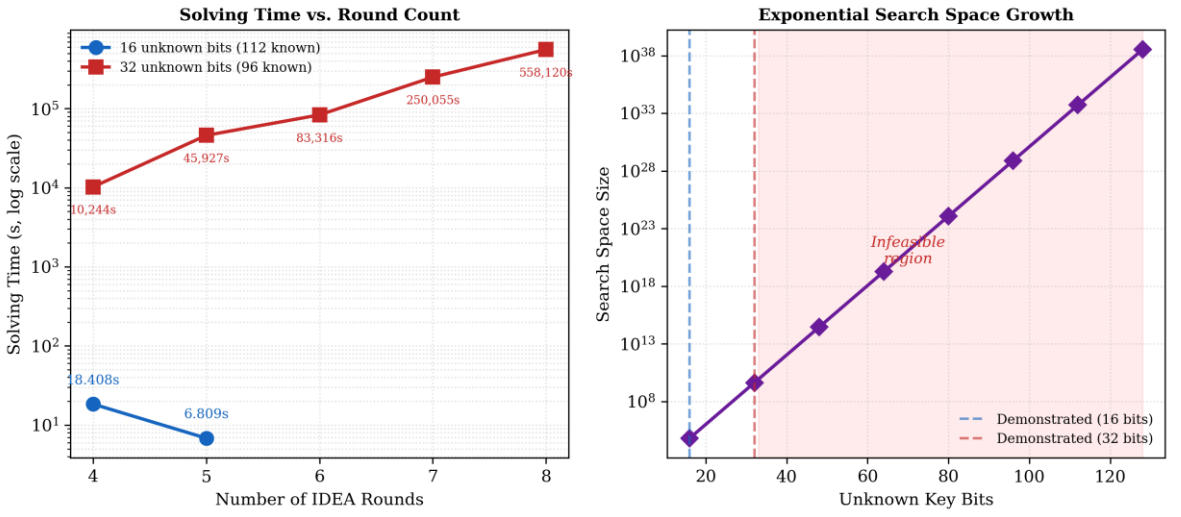

Fig. 10: Left: solving time versus round count for both partial-knowledge scenarios. Right: exponential growth of the brute-force search space as the number of unknown key bits increases.

An exhaustive search over 16 unknown bits contains 65,536 candidates; 32 unknown bits contain 4,294,967,296. The purpose of the experiment is not to claim that Z3 dominates a tuned brute-force loop. It provides one uniform model that accepts arbitrary disclosure patterns and verifies returned models automatically. Comparisons with the archival series remain indicative because the original logs are missing.

The appropriate conclusion is limited. Under the stated conditions, reduced-round instances with substantial disclosure are tractable; the data do not demonstrate a vulnerability in a full-round IDEA or its key schedule. The archival study should be rerun with raw logs, several keys, and controlled solver settings before broader claims are made. Full-round encoding remains an open direction.

6. Conclusion

This study examined whether word-level QF_BV reasoning through Z3py can recover key information from reduced-round IDEA in a measurable and independently verifiable setting. Algebraic constraints complement, rather than replace, differential and linear analyses. The resulting model represents multiplication modulo 2^16+1 with its zero-operand convention, the 25-bit key schedule, the round function, and the output transformation. A documented architecture and workflow surround a CSV-backed multi-seed campaign, recovering the remaining 16 bits when 112 are disclosed. Unconstrained search was also tested, but a single pair does not generally make full-key recovery unique. For the 16-unknown-bit campaign, the median times range from 62.01 s to 165.1 s over the tested round counts. Each solved model matched the planted key. The principal outcome is a reproducible word-level constraint benchmark with explicit verification, not an asymptotic advantage over enumeration of small key spaces. The measurements cover a reduced-round IDEA, one pair, one planted key, and one hardware and solver environment. Partial-key recovery is the established baseline; unconstrained models are limited by the number of keys compatible with one pair. The archival 96-bit series require fresh execution before statistical or security claims can rely on it. Future work should extend the model to deeper and full-round instances, assess incremental key-bit constraints and parallel solving, and translate related Lai-Massey or ARX designs into QF_BV. Other priorities include noisy side-channel constraints, comparisons with CVC5 and Boolector, and a multi-key, multi-pair study of the full-key tractability frontier.

CRediT Author Contribution Statement

Praveen Kumar Gundaram: Conceptualization, Formal analysis, Investigation, Methodology, Software, Validation, Writing – Original draft, Writing – Review & editing. The author has read and agreed to the published version of the manuscript.

Data Availability Statement

The package contains the Python solver and integer reference implementation, the parallel sweep driver, generated figures, results.csv, and the test-vector data used for the measured cells. The archival 96-known-bit timings are retained for traceability but have no raw solver logs. All supporting materials are available from the corresponding author upon reasonable request.

Funding Declaration

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

Conflict of Interest

There is no conflict of interest.

Artificial Intelligence (AI) Use Disclosure

AI-assisted tools were used for language editing, consistency checks, and document assembly. The author remains responsible for the scientific content, validation of the reported results, and the final manuscript. No AI-generated image is used as experimental evidence.

Supporting information

Not Applicable

References

[1] C. E. Shannon, Communication theory of secrecy systems, *Bell System Technical Journal*, 1949, **28**, 656–715, 1949, doi: 10.1002/j.1538-7305.1949.tb00928.x.

[2] E. Biham, A. Shamir, Differential cryptanalysis of DES-like cryptosystems, *Journal of Cryptology*, 1991, **4**, 3-72, doi: 10.1007/BF00630563.

[3] M. Matsui, Linear cryptanalysis method for DES cipher, in: Helleseth, T. (eds) Advances in Cryptology- EUROCRYPT '93. EUROCRYPT 1993. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 1994, **765**, 386–397, doi: 10.1007/3-540-48285-7_33.

[4] N. T. Courtois, J. Pieprzyk, Cryptanalysis of block ciphers with overdefined systems of equations, in Y. Zheng, (eds) Advances in Cryptology - ASIACRYPT 2002. ASIACRYPT 2002. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2002, 2501, 267–287, doi: 10.1007/3-540-36178-2_17.

[5] G. V. Bard, Algebraic cryptanalysis, Springer New York, NY, 2009, doi: 10.1007/978-0-387-88757-9.

[6] X. Lai, J. L. Massey, A proposal for a new block encryption standard, In: I. B. Damgård, (eds) Advances in Cryptology - EUROCRYPT '90, EUROCRYPT 1990, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 1991, 473, 389–404, doi: 10.1007/3-540-46877-3_35.

[7] J. Daemen, R. Govaerts, J. Vandewalle, Weak Keys for IDEA, (eds) Advances in Cryptology - CRYPTO' 93. CRYPTO 1993, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 1994, 773, 224–231, doi: 10.1007/3-540-48329-2_20.

[8] D. Khovratovich, G. Leurent, C. Rechberger, Narrow-bicliques: cryptanalysis of full IDEA, (eds) Advances in Cryptology – EUROCRYPT 2012. EUROCRYPT 2012, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2012, 7237, 392–410, doi: 10.1007/978-3-642-29011-4_24.

[9] C. Barrett, A. Stump, C. Tinelli, The SMT-LIB Standard: Version 2.0, in Proceedings of SMT Workshop, 2010.

[10] F.-X. Standaert, G. Piret, J.-J. Quisquater, Cryptanalysis of block ciphers: a survey, UCL Crypto Group Technical Report, 2003.

[11] M. Soos, K. Nohl, C. Castelluccia, Extending SAT Solvers to Cryptographic Problems. In: Kullmann, O. (eds) Theory and Applications of Satisfiability Testing - SAT 2009. SAT 2009. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2009, 5584, 244–257, doi: 10.1007/978-3-642-02777-2_24.

[12] H. K. Sahu, N. R. Pillai, I. Gupta, R. K. Sharma, SMT solver-based cryptanalysis of block ciphers, *SN Computer Science*, 2020, **1**, 169, doi: 10.1007/s42979-020-00181-4.

[13] P. K. Gundaram, A. Naidu Tentu and N. B. Muppalaneni, Performance of various SMT Solvers in Cryptanalysis, 2021 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS), Greater Noida, India, 2021, 298-303, doi: 10.1109/ICCCIS51004.2021.9397110.

[14] L. de Moura, N. Bjørner, Z3: an efficient SMT solver, In: C. R. Ramakrishnan, J. Rehof, (eds) Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2008. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2008, 4963, 337–340, doi: 10.1007/978-3-540-78800-3_24.

[15] R.-P. Weinmann, Algebraic methods in block cipher cryptanalysis, PhD thesis, TU Darmstadt, 2009.

[16] G. V. Bard, N. T. Courtois, C. Jefferson, Efficient methods for conversion and solution of sparse systems of low-degree multivariate polynomials over GF(2) via SAT-solvers, IACR Cryptology ePrint Archive, Report 2007/024, 2007.

[17] N. T. Courtois, G. V. Bard, Algebraic cryptanalysis of the data encryption standard, in: S. D. Galbraith (ed.), Cryptography and Coding 2007, Lecture Notes in Computer Science, 4887, Springer, Berlin, Heidelberg, 2007, pp. 152–169, doi: 10.1007/978-3-540-77272-9_10.

[18] C. Cid, R.-P. Weinmann, Block Ciphers: Algebraic Cryptanalysis and Gröbner Bases. In: Sala, M., Sakata, S., Mora, T., Traverso, C., Perret, L. (eds) Gröbner Bases, Coding, and Cryptography. Springer, Berlin, Heidelberg, 2009, 307–327, doi: 10.1007/978-3-540-93806-4_17.

[19] F. Lafitte, J. Nakahara Jr., D. Van Hamme, Applications of SAT solvers in cryptanalysis: finding weak keys and preimages, *Journal of Satisfiability, Boolean Modeling and Computation*, 2014, **9**, 1–25, doi: 10.3233/SAT190099.

[20] J. Chen, D. Xue, X. Lai, An analysis of international

data encryption algorithm (IDEA) security against differential cryptanalysis, *Wuhan University Journal of Natural Sciences*, 2008, **13**, 697–701, doi: 10.1007/s11859-008-0612-4.

Publisher Note: The views, statements, and data in all publications solely belong to the authors and contributors. GR Scholastic is not responsible for any injury resulting from the ideas, methods, or products mentioned. GR Scholastic remains neutral regarding jurisdictional claims in published maps and institutional affiliations.