Solving AES-SAT Using Side-Channel Hints: A Practical Assessment

Elena Dubrova
KTH Royal Institute of Technology
Stockholm, Sweden
Email: dubrova@kth.se

***Abstract*—Side-channel attacks exploit information leaked through non-primary channels, such as power consumption, electromagnetic emissions, or timing, to extract sensitive data from cryptographic devices. Over the past three decades, side-channel analysis has evolved into a mature research field with well-established methodologies for analyzing standard cryptographic algorithms like the Advanced Encryption Standard (AES). However, the integration of side-channel analysis with formal methods remains relatively unexplored. In this paper, we present a hybrid attack on AES that combines side-channel analysis with SAT. We model AES as a SAT problem and leverage hints of the input and output values of the S-boxes, extracted via profiled deep learning-based power analysis, to solve it. Experimental results on an ATXmega128D4 MCU implementation of AES-128 demonstrate that the SAT-assisted approach consistently recovers the full encryption key from a single trace, captured from devices different from those used for profiling, within one hour. In contrast, without SAT assistance, the success rate remains below 80% after 26 hours of key enumeration.**

I. Introduction

SAT solvers are software tools specifically designed to address the Boolean satisfiability (SAT) problem [1]. By utilizing various heuristics for variable and value selection, clause learning, restart, backtracking, etc., they navigate the exponential search space of possible values of variables in search of a satisfying assignment. Despite the NP-completeness of the SAT problem [1], contemporary SAT solvers can typically handle problem instances involving hundreds of variables. Practical applications of SAT solvers include formal verification [2], automatic test pattern generation [3], and logic synthesis [4].

SAT solvers have been used in cryptanalysis in the past, including finding preimages and collisions for hash functions [5], [6], forging RSA signatures [7], identifying weak keys [8] and short cycles [9], guess-and-determine attacks [10], and algebraic attacks [11]–[13]. The concept of *logical cryptanalysis*, introduced by Massacci and Marraro [14], provided a general framework for encoding the low-level properties of cryptographic algorithms as a SAT instance in order to analyze them using SAT solvers and other formal methods such as theorem proving. This methodology has been followed in [15]–[17]. Nonetheless, mounting a practical attack on a full, non-reduced version of a (secure) cryptographic algorithms is typically difficult.

The idea to use SAT solvers an their variants in physical attacks on implementations of cryptogrphic algorithms was first explored in the context of cold boot attacks [18]–[22]. A cold boot attack extracts data from a non-powered random access memory (RAM) by cooling down the RAM [23]. The cooling significantly increases the data retention time, e.g. to 10 hours at -50°C as opposed to 10 seconds at the room temperature for some static RAMs [24]. SAT-assisted cold boot attacks attempts to recover secret keys from from decayed RAM images of expanded round keys by exploiting redundancy of the key scheduling algorithms.

Potlapally et al. [25] first proposed the use of SAT solvers in side-channel attacks on software implementations of cryptographic algorithms. Side-channel attacks exploit information extracted through physically measurable, non-primary channels such as timing [26], power [27], or electromagnetic radiation [28] emitted by the device running the algorithm. The recovered intermediate variables are used to constrain values of the corresponding literals in the SAT representation of the algorithm, thus making the SAT problem easier to solve. Using simulated data, it is demonstrated in [29] that secret keys of DES and 3DES can be extracted from their software implementations using a SAT-assisted side-channel attack, provided the intermediate variables recovered via side channels are separated by fewer than five DES rounds. Similar ideas were explored using simulated data in [30]–[32].

An essential difference between a SAT-assisted cold boot attack and a SAT-assisted side-channel attack lies in the scope of the SAT modeling. In the former, only the key schedule needs to be modeled as a SAT problem, whereas in the latter, the entire cryptographic algorithm must be described. Consequently, the model becomes significantly larger, making the SAT problem harder to solve.

Contributions: This paper demonstrates a practical attack on an ATXmega128D4 MCU implementation of AES-128 that combines profiled deep learning-based side-channel analysis with SAT.

We first model AES-128 as a SAT problem using 9,152 literals and 188,664 clauses. Then we simplify the SAT problem by adding clauses that incorporate hints of the input and output values of the S-boxes extracted via power analysis. Unlike traditional power analysis, which typically uses S-boxes in the first or the last round of AES, the SAT-assisted power analysis can utilize S-boxes in any intermediate round. Our experiments show that the SAT-assisted approach can consistently recover the full encryption key from a single power trace, captured

from devices different from those used for profiling, within one hour. Without SAT assistance, the success rate is at most 62.6% within one hour and at most 79.3% after 26 hours of key enumeration.

We would like to stress the importance of using real rather than simulated data, employing different devices for profiling and attacks, and recovering the full key for a comprehensive evaluation of side-channel attacks.

Simulated data often fail to account for factors such as temperature fluctuations, power supply instability, limitations in the precision and resolution of measurement instruments, correlated noise, or imperfect timing synchronization. These factors can make it significantly harder to extract exploitable side-channel information from actual devices. Thus, studies relying solely on simulation may overstate the ease with which cryptographic implementations can be compromised in practice.

Similarly, studies that use the same physical device for both profiling and attacks may reach overly optimistic conclusions, as they neglect inter-chip variations that cause side-channel profiles to be device-specific.

Finally, studies that recover only a part of the key and assume the remaining parts can be recovered with the same probability may draw incorrect conclusions. The difficulty of recovering different subkeys can vary significantly.

The rest of the paper is organized as follows. Section II provides the background information. Section III describes the adversary model. Section IV present the SAT-assisted side-channel analysis method. Section V describes the experimental results. Section VI concludes the paper and discusses future work.

II. Background

A. AES algorithm

AES [33] is a symmetric encryption algorithm standardized by the NIST in FIPS 197 and included in ISO/IEC 18033-3. AES takes a 128-bit block of plaintext and an n-bit key, K, as input, and produces a 128-bit block of ciphertext as output. AES supports key sizes of $n = 128$, 192, and 256 bits, with 128 bits being the most commonly used. In this paper, we focus on this case, referred to as AES-128.

The number of rounds in AES depends on the key size. AES with a 128-bit, 192-bit, or 256-bit key performs $k = 10$, 12, and 14 rounds, respectively. Each round, except the last one, executes the following four steps:

1) Non-linear substitution, SubBytes,
2) Transposition of rows, ShiftRows,
3) Mixing of columns, MixColumns,
4) Round key addition, AddRoundKey,

Each round uses a different round key, RKi, for $i \in 1, \ldots, k$, derived from the original key K. The last round skips the MixColumns step.

Like any block cipher, AES can be used in several modes of operation. In this paper, we assume the electronic codebook (ECB) mode, where the message is divided into blocks and each block is encrypted separately.

B. The Boolean satisfiability problem

1) Problem formulation: The Boolean satisfiability (SAT) problem [34] is defined as follows: Given a propositional Boolean formula, decide if there exists an assignment of true or false values of the variables such that the formula evaluates to true. If such an assignment exists, the formula is satisfiable; otherwise, it is unsatisfiable. For a Boolean formula with n variables, there are 2^n possible assignments of variables.

To represent the SAT problem, propositional Boolean formulas are typically written in Conjunctive Normal Form (CNF). In a CNF, each variable symbol, x or $\overline{x}$, is called a *literal*. A *clause* is a disjunction (Boolean OR) of literals, and the CNF itself is a conjunction (Boolean AND) of clauses.

2) SAT Solvers: While the SAT problem is known to be NP-complete [1], contemporary SAT solvers can efficiently handle SAT instances containing hundreds of variables. Most of today's SAT solvers are based on the Davis-Putnam-Logemann-Loveland (DPLL) algorithm [35], an improvement of the Davis-Putnam algorithm [36]. Well-known DPLL-based SAT solvers include GRASP [37], which introduced a robust conflict analysis method, and Chaff [38], known for its efficient Boolean constraint propagation and novel decision strategy with minimal overhead. SAT solvers that utilize these strategies are referred to as *conflict-driven*.

In our experiments, we use MiniSat [39], a conflict-driven SAT solver recognized for its adaptability across various domains. We selected MiniSat because of its competitive performance, open-source availability, and its ability to incorporate arbitrary Boolean constraints.

3) MiniSat: MiniSat [39] employs a depth-first search algorithm with backtracking to find a satisfying assignment of variables for a given set of clauses. The algorithm heuristically selects a variable, guesses its value as true or false, and evaluates whether the values of other variables are influenced by this assumption. Based on this assessment, the affected variables are assigned, and the search continues until further assignments become unfeasible.

During the propagation phase, the algorithm may encounter a conflict, meaning that a clause is found to be unsatisfiable due to all its literals being false. In such cases, a *learned clause* is generated and added to the SAT problem. This clause describes the erroneous guess that led to the conflict.

To resolve conflicts, the algorithm backtracks until the learned clause simplifies to a unit clause. Once this is achieved, the clause is propagated, and the search process resumes. The learned clauses guide the backtracking process, helping the algorithm make more informed guesses and accelerating subsequent conflict resolutions by identifying the root cause of the conflict.

Eventually, MiniSat either finds a satisfiable assignment or determines that no such assignment exists.

4) DIMACS Format: The DIMACS format is a standard format for representing a propositional Boolean formula in CNF. Many SAT solvers, including MiniSat, use DIMACS as a standard interface.

A file in the DIMACS format starts with a header line of the form `p cnf <variables> <clauses>`, where `<variables>` and `<clauses>` represent the number of variables and clauses in the CNF, respectively. Each line that begins with the `c` character is interpreted as a comment. Lines that do not begin with `p` or `c` are interpreted as CNF clauses consisting of the Boolean OR of literals.

To describe literals x and $\overline{x}$, a unique integer is assigned to each variable of the CNF. A positive integer represents x and a negative one represents $\overline{x}$. The symbol 0 is used to denote the end of a clause. The Boolean AND of all the clauses in the file defines the CNF.

For example, a CNF for the two-variable Boolean XOR function $a \oplus b = c$ is represented by the following four clauses:

$$\begin{gathered} \neg a \wedge \neg b \wedge \neg c \\ \neg a \wedge b \wedge c \\ a \wedge \neg b \wedge c \\ a \wedge b \wedge \neg c \end{gathered}$$

Note that both cases, when the function evaluates to 1 and to 0, are included in the CNF. The corresponding DIMACS representation, assuming the variables are assigned as $a = 1, b = 2, c = 3$ is:

```
p cnf 3 4
-1 -2 -3 0
-1  2  3 0
 1 -2  3 0
 1  2 -3 0
```

C. Profiled side-channel attacks

Side-channel attacks can be classified into two main types: profiled and non-profiled attacks.

Profiled attacks first model the information leakage of the target implementation using one or more devices similar to the device under attack, called *profiling* devices. The modeling can be done either by creating templates [40], or by training neural networks [41]. The resulting models are then used to recover the secret variable (e.g., the key) from the cryptographic algorithm executed on the device under attack.

Non-profiled attacks, on the other hand, are applied directly, without the need to model the leakage of the target implementation beforehand [42]. These attacks may involve statistical methods such as correlation power analysis [43], or unsupervised machine learning algorithms such as k-means.

III. Adversary model

This section defines the three main components of an adversary model in accordance with [44]: adversary goals, assumptions, and capabilities.

Assumptions: We assume that the adversary has physical access to the device under attack which runs the AES algorithm. Additionally, the adversary is assumed to have fully controllable profiling devices that are similar to the device under attack.

Capabilities: The adversary is equipped with power analysis tools and has expertise in side-channel attacks, AES, and deep learning. The adversary is capable of capturing power traces from the device under attack during the execution of the AES algorithm.

Goals: The adversary's goal is to obtain the encryption key of AES. To achieve this, the adversary first attempts to extract partial information about intermediate variables from the implementation of AES running on the device under attack via side-channel analysis. The adversary then applies SAT techniques to recover the full key.

IV. SAT-assisted side-channel attack method

A. Expressing AES in CNF

Expressing AES as a SAT problem involves representing its four main steps—SubBytes, ShiftRows, MixColumns, AddRoundKey—along with the key schedule algorithm in CNF. This requires breaking down each operation into Boolean functions defined in terms of binary variables and then converting the resulting sum-of-products expressions of these functions into CNF clauses.

a) SubBytes: The SubBytes step substitutes each byte in the AES state with another byte using the AES *S-box* permutation, which is a Boolean function of type $f_{\textit{S-box}} : \{0,1\}^8 \to \{0,1\}^8$. A logic minimizer such as Espresso can be used to find a minimal sum-of-products form of $f_{\textit{S-box}}$, which is then translated into CNF clauses.

b) ShiftRows: The ShiftRows step cyclically shifts the bytes in each row of the 4×4 AES state matrix. Each byte in the ith row is shifted i positions to the left, for $i \in \{0, 1, 2, 3\}$. To encode this operation into CNF, the mapping of the bytes from their original positions to their shifted positions is translated into CNF clauses.

c) MixColumns: The MixColumns step combines the four bytes of each column of the 4×4 AES state matrix using an invertible linear transformation involving a matrix multiplication over GF(2^8), modulo the irreducible polynomial $x^8 + x^4 + x^3 + x^1$. The Boolean function defining MixColumns is of type $\{0,1\}^{32} \to \{0,1\}^{32}$ since each of the four input bytes affects all four output bytes. To represent this in CNF, auxiliary variables are typically introduced to split complex functions into smaller ones.

d) AddRoundKey: The AddRoundKey step performs a bitwise XOR between the AES state and the round key. To convert this operation into CNF, each XOR operation between corresponding bits of the state matrix and the round key is expressed as CNF clauses.

e) Key Schedule Algorithm: The key schedule algorithm generates round keys $RK_1, \ldots, RK_k$ from the original key K. This process involves a combination of byte substitutions using the S-box, one-byte left circular shifts, XOR operations, and the application of round constants.

To convert the key schedule algorithm into CNF, binary variables representing the key K and round keys $RK_1, \ldots, RK_k$ are first introduced. The S-box substitution used in the key expansion is encoded in the same way as the SubBytes step, through CNF clauses derived from its minimal sum-of-products representation. The cyclic shifts of bytes are defined using binary relationships between input and output

TABLE I: Profiling stage duration.

Time to capture 10K traces	# Devices	Total time
1 hr	2	2 hrs

Time to train one NN	# NNs	Total time
3 hrs	10	30 hrs

positions, similar to the ShiftRows encoding. Finally, the XOR operations combining intermediate states and round constants are expressed and converted into CNF.

There are tools available for automating CNF generation. For instance, we used `https://github.com/meelgroup/aes-cnf-gen/`.

B. Equipment

In the experiments, we use a software implementation of AES-128 running on an ATXmega128D4 MCU. Two devices are used for profiling, while three other devices are used for the attacks. Power consumption is measured using a ChipWhisperer-Lite, which also controls the communication between the device and the computer.

C. Profiling details

At the profiling stage, we capture 10,000 power traces from each profiling device for neural network training. Each trace is captured during the execution of the AES encryption for messages and keys selected at random. We then identify the segments corresponding to each S-box in every round.

We extract the segments for all S-boxes from the rounds, starting from the first, R_1, to the ninth, R_9, and take their union. This increases the total size of the training set to 20,000 $\times\ 9 \times 16 = 2,880,000$. The last round is not included because its traces differ from those of other rounds[1].

Table I shows that it takes 2 hours to capture 20,000 traces. In contrast, it would require 12 days to capture a training set containing 2.88 million traces without applying the expansion technique described above.

Using the expanded training set, we train five neural networks using the S-box input byte value as a label, and five neural networks using the S-box output byte value as a label. In all cases, a multilayer perceptron (MLP) architecture with three dense layers is used.

From Table I, one can see that training one neural network takes 3 hours on a PC with an Intel Core i7-1370P CPU and 64GB of RAM. As a result, the total profiling time amounts to 32 hours.

V. Experimental Results

A. Key recovery by power analysis only

For this experiment, we capture from each device under attack 1,000 power traces during the execution of the AES encryption for random messages and a fixed key (different for each device). Using ensembles consisting of five neural network models trained during the profiling stage, we recover S-box input and output byte values in rounds R_1-R_9. The predictions from individual models are aggregated by computing the cumulative probability of their score vectors.

Table II lists the empirical probabilities of recovering an S-box input value from a single trace in rounds R_1 to R_9. Each probability is computed as the mean over 1,000 tests. Table III presents similar results for the S-box outputs.

From Tables II and III we can see that success probabilities vary among different devices. This may due to differences in physical characteristics of device's components such as transistors, capacitors, etc, caused by manufacturing process variation. This may also due to differences in device age of wear-off level. Hardware aging and wear-off effects such as transistor degradation or increased leakage current can affect power consumption profile of a device.

Next, we quantify the likelihood of recovering the full key from a single trace using power analysis alone. It is known that such attacks are feasible for the ATXmega128D4 MCU implementation of AES-128 [45]. An attacker typically employs a divide-and-conquer strategy with a subkey size of one byte, focusing on the S-boxes in either the first or the last round of AES, as S-boxes in other rounds depend on multiple key bytes.

Table IV shows the empirical probability to recover at least n out of the 16 possible S-box input/output values in the first round from a single trace. We can see that the probability of recovering all 16 bytes is at most 14.4%. If the attack is unsuccessful, the attacker may attempt to search through the $\binom{16}{k}$ possible combinations of k out of 16 bytes and enumerate the 2^{8k} possible subkeys. The last column of Table IV shows the time required for such an enumeration, calculated based on the estimate that 100,000 encryptions can be performed per second. We can see that the probability of recovering the full key is at most 79.3% after 26.1 hours of enumeration.

B. Key recovery by SAT-assisted power analysis

The SAT-assisted side-channel analysis can utilize S-box input and output values in any round of AES. Table V shows the likelihood of recovering at least n of out of 144 possible S-box input/output values in rounds R_1-R_9 from a single trace. The same set of traces as in the tests of Table IV is used. The fourth column shows the SAT solver runtime for the case when n S-box input and output values in rounds R_1-R_9 are known. Each entry represents the mean of 100 tests in which the positions of known S-boxes are selected at random. We can see that, if 60 or more S-box input and output values are known, finding a satisfying assignment takes less than one second. The likelihood of this is 0.831/0.784 for S-box inputs/outputs.

Finally, we perform 30 end-to-end SAT-assisted single-trace attacks (10 for each target device). In each case, we first predict all 144 S-box input and output values in rounds R_1-R_9 using ensembles of five models trained at the profiling stage, Then, we sort the predicted bytes in the descending order based

[1]Since the last round does not mix columns, in the C implementation of AES, its code is outside the for-loop executing other rounds. Consequently, a compiler may optimize it differently.

TABLE II: The empirical probability to recover an S-box input value from a single trace (mean for 1,000 tests).

Device	Round									Avg.
	1	2	3	4	5	6	7	8	9	
D_1	0.913	0.921	0.934	0.941	0.953	0.938	0.939	0.934	0.947	0.936
D_2	0.783	0.968	0.975	0.979	0.887	0.965	0.977	0.970	0.978	0.943
D_3	0.853	0.822	0.847	0.870	0.892	0.878	0.888	0.875	0.895	0.869

TABLE III: The empirical probability to recover an S-box output value from a single trace (mean for 1,000 tests).

Device	Round									Avg.
	1	2	3	4	5	6	7	8	9	
D_1	0.910	0.917	0.931	0.941	0.951	0.936	0.933	0.933	0.946	0.933
D_2	0.790	0.967	0.973	0.978	0.892	0.961	0.974	0.969	0.976	0.942
D_3	0.852	0.812	0.833	0.861	0.885	0.869	0.876	0.868	0.889	0.861

TABLE IV: The empirical probability to recover at least n out of 16 S-box input/output values in R_1 from a single trace.

n	Mean probability to recover $\geq n$ bytes in R_1		Estimated time for key enumeration
	inputs	outputs	
12	0.884	0.885	1.89 years
13	0.793	0.790	26.1 hrs
14	0.626	0.626	1.31 min
15	0.393	0.383	0.04 sec
16	0.142	0.144	0 sec

TABLE V: The empirical probability to recover at least n out of 144 S-box input/output values in R_1-R_9 from a single trace.

n	Mean probability to recover $\geq n$ bytes in R_1-R_9		Mean SAT solver runtime for n known bytes	# Timeouts for 1 hour timeout
	inputs	outputs		
40	0.946	0.917	19.9 min	31
50	0.898	0.857	5.25 min	5
60	0.831	0.784	0.68 sec	0
70	0.744	0.693	0.24 sec	0

TABLE VI: The results of SAT-assisted single trace attacks using S-boxes in R_1-R_9. The SAT solver timeout is 1 min.

Rounds	% Recovered keys	Mean time for full key recovery	Guess range
R_1	60%	53.3 sec	[14:16]
R_1-R_2	80%	99.1 sec	[15:25]
R_1-R_3	90%	5.7 min	(20:30]
R_1-R_4	90%	11.3 min	(25:35]
R_1-R_5	90%	17.6 min	(30:40]
R_1-R_6	90%	14.6 min	(40:50]
R_1-R_7	96.7%	6.85 min	(45:55]
R_1-R_8	96.7%	20.2 min	(50:60]
R_1-R_9	96.7%	20.2 min	(55:65]

on the probabilities of their score vectors. We add the top m predictions as clauses constraining S-boxes in the CNF of the AES and run MiniSAT solver [39] to find a satisfying assignment within a given time. If either an UNSAT is returned, or a timeout is reached, m is decremented and the attack is repeated.

Table VI summarizes the results. The "guess range" column indicates the number m of top predictions used to constrain the CNF. The results show that SAT-assisted side-channel analysis can recover the full key from a single power trace in 96.7% of cases (29 out of 30) within a one-minute SAT solver runtime. Extending the timeout to one hour allows recovery of the remaining key, resulting in a total attack time of 8.9 hours. This is a significant improvement over pure power analysis, where the success rates for one-minute and one-hour key enumeration time are at most 39.3% and 62.6%, respectively (see Table IV).

VI. Conclusion

We presented an attack on AES that combines profiled deep learning-based side-channel analysis with SAT. To the best of our knowledge, this is the first work to comprehensively evaluate the difficulty of solving the AES-SAT problem using real side-channel data, incorporating different devices for profiling and attacks, and successfully recovering the full key.

Our experimental results demonstrate that integration of a SAT solver is particularly advantageous when the device under attack has aging or wear-out characteristics different from the profiling devices. Information about these characteristics is typically unavailable in a real attack scenario.

Future work includes investigating the possibility of more efficient modeling of AES as an MV-SAT problem [46], applying advanced spectral techniques [47] to side-channel analysis, and exploring the use of additional redundant values as a potential countermeasure against side-channel attacks.

VII. Acknowledgments

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation and the Swedish Civil Contingencies Agency (Grant No. 2020-11632).

References

[1] S. A. Cook, "The complexity of theorem-proving procedures," in *Proceedings of the Third Annual ACM Symposium on Theory of Computing (STOC '71)*, 1971, pp. 151–158.

[2] P. Abdula, P. Bjesse, and N. Een, "Symbolic reachability analysis based on SAT-solvers," in *Proc. of the 6th International Conference on Tools and Algorithms for the Analysis and Construction of Systems (TACAS'2000)*, vol. 1785 of LNCS. Springer-Verlag, 2000.

[3] W. Kunz, J. Marques-Silva, and S. Malik, *Sat and ATPG: Algorithms for Boolean Decision Problems*. Boston, MA: Springer US, 2002, pp. 309–341. [Online]. Available: https://doi.org/10.1007/978-1-4615-0817-5_12

[4] S. Hassoun and S. Tsutomu, *Logic Synthesis and Verification*. Norwell, MA, USA: Kluwer Academic Publishers, 2002.

[5] I. Mironov and L. Zhang, "Applications of SAT solvers to cryptanalysis of hash functions," in *Proc. of the 9th International Conference on Theory and Applications of Satisfiability Testing*, ser. SAT'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 102–115. [Online]. Available: http://dx.doi.org/10.1007/11814948_13

[6] P. Morawiecki and M. Srebrny, "A SAT-based preimage analysis of reduced Keccak hash functions," *Inf. Process. Lett.*, vol. 113.10-11, pp. 392–397, 2013.

[7] C. Fiorini, E. Martinelli, and F. Massacci, "How to fake an RSA signature by encoding modular root finding as a SAT problem," *Discrete Appl. Mat*.

[8] F. Lafitte, J. Nakahara, and V. D. Heule, "Applications of SAT solvers in cryptanalysis: Finding weak keys and preimages," *Journal of Politics in Latin America;2014, Vol. 6 Issue 2, p1*, vol. 6, 2014.

[9] E. Dubrova and M. Teslenko, "An efficient SAT-based algorithm for finding short cycles in cryptographic algorithms," in *IEEE Int. Symposium on Hardware-Oriented Security and Trust*, 2018.

[10] F. Lafitte, O. Markowitch, and D. Van Heule, "SAT based analysis of LTE stream cipher ZUC," in *Proc. of the 6th International Conference on Security of Information and Networks*, ser. SIN '13. New York, NY, USA: ACM, 2013, pp. 110–116. [Online]. Available: http://doi.acm.org/10.1145/2523514.2523533

[11] B. Chen, "Strategies on algebraic attacks using SAT solvers," in *The 9th International Conference for Young Computer Scientists (ICYCS'2008)*, Nov 2008, pp. 2204–2209.

[12] M. Soos, K. Nohl, and C. Castelluccia, "Extending SAT solvers to cryptographic problems," in *Proc. of the 12th Int. Conf. on Theory and Applications of Satisfiability Testing*, ser. SAT '09. Springer-Verlag, 2009, pp. 244–257.

[13] P. Jovanovic and M. Kreuzer, "Algebraic attacks using SAT-solvers," *Groups Complexity Cryptology*, vol. 2.2, pp. 247–259, 2010.

[14] F. Massacci and L. Marraro, "Logical cryptanalysis as a SAT problem," *J. Autom. Reason.*, vol. 24, no. 1-2, pp. 165–203, Feb. 2000.

[15] D. Jovanović and P. Janičić, "Logical analysis of hash functions," in *Proc. of the 5th International Conference on Frontiers of Combining Systems*, ser. FroCoS'05. Springer-Verlag, 2005, pp. 200–215.

[16] T. Eibach, E. Pilz, and G. Völkel, *Attacking Bivium Using SAT Solvers*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 63–76.

[17] E. Homsirikamol, P. Morawiecki, M. Rogawski, and M. Srebrny, *Security Margin Evaluation of SHA-3 Contest Finalists through SAT-Based Attacks*. Springer Berlin Heidelberg, 2012, pp. 56–67.

[18] A. A. Kamal and A. M. Youssef, "Applications of SAT solvers to AES key recovery from decayed key schedule images," in *Int. Conf. on Emerging Security Information, Systems and Technologies*, 2010, pp. 216–220.

[19] A. Tsow, "An improved recovery algorithm for decayed AES key schedule images," in *International Workshop on Selected Areas in Cryptography*. Springer, 2009, pp. 215–230.

[20] X. Liao, H. Zhang, M. Koshimura, H. Fujita, and R. Hasegawa, "Using MaxSat to correct errors in AES key schedule images," in *2013 IEEE 25th international conference on tools with artificial intelligence*. IEEE, 2013, pp. 284–291.

[21] T. Tanigaki and N. Kunihiro, "Maximum likelihood-based key recovery algorithm from decayed key schedules," in *ICISC 2015*. Springer, 2015, pp. 314–328.

[22] I. Zimerman, E. Nachmani, and L. Wolf, "Recovering AES keys with a deep cold boot attack," ArXiv, 2106.04876, 2021.

[23] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten, "Lest we remember: Cold-boot attacks on encryption keys," *Commun. ACM*, vol. 52, no. 5, p. 91–98, may 2009. [Online]. Available: https://doi.org/10.1145/1506409.1506429

[24] S. Skorobogatov, "Physical attacks on tamper resistance: Progress and lessons," in *Proceedings of the 2nd ARO Special Workshop on HW Assurance*, 2011.

[25] N. Potlapally, A. Raghunathan, S. Ravi, N. Jha, and R. Lee, "Satisfiability-based framework for enabling side-channel attacks on cryptographic software," in *Proc. of the DATE*, vol. 2, 2006, pp. 1–6.

[26] P. C. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems," in *Advances in Cryptology — CRYPTO '96*, N. Koblitz, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 104–113.

[27] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Advances in Cryptology — CRYPTO' 99*, M. Wiener, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 388–397.

[28] K. Gandolfi, C. Mourtel, and F. Olivier, "Electromagnetic analysis: Concrete results," in *International workshop on cryptographic hardware and embedded systems*. Springer, 2001, pp. 251–261.

[29] N. R. Potlapally, A. Raghunathan, S. Ravi, N. K. Jha, and R. B. Lee, "Satisfiability-based framework for enabling side-channel attacks on cryptographic software," in *Proc. of the Conference on Design, Automation and Test in Europe: Designers' Forum*, ser. DATE '06. 3001 Leuven, Belgium, Belgium: European Design and Automation Association, 2006, pp. 18–23. [Online]. Available: http://dl.acm.org/citation.cfm?id=1131355.1131360

[30] M. Renauld, F.-X. Standaert, and N. Veyrat-Charvillon, "Algebraic side-channel attacks on the AES: Why time also matters in DPA," in *Cryptographic Hardware and Embedded Systems – CHES 2009*, ser. Lecture Notes in Computer Science, C. Clavier and K. Gaj, Eds., vol. 5747. Springer, 2009, pp. 97–111.

[31] Y. Oren, M. Kirschbaum, T. Popp, and A. Wool, "Algebraic side-channel analysis in the presence of errors," in *Cryptographic Hardware and Embedded Systems, CHES 2010*, S. Mangard and F.-X. Standaert, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 428–442.

[32] Y. Oren and A. Wool, "Tolerant algebraic side-channel analysis of AES," Cryptology ePrint Archive, Paper 2012/092, 2012. [Online]. Available: https://eprint.iacr.org/2012/092

[33] J. Daemen and V. Rijmen, *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer, 2002.

[34] M. R. Garey and D. S. Johnson, *A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[35] M. Davis, G. Logemann, and D. Loveland, "A machine program for theorem-proving," *Commun. ACM*, vol. 5, no. 7, pp. 394–397, Jul. 1962. [Online]. Available: http://doi.acm.org/10.1145/368273.368557

[36] M. Davis and H. Putnam, "A computing procedure for quantification theory," *J. ACM*, vol. 7, no. 3, pp. 201–215, Jul. 1960. [Online]. Available: http://doi.acm.org/10.1145/321033.321034

[37] J. P. M. Silva and K. A. Sakallah, "Conflict analysis in search algorithms for satisfiability," in *Proc. Eighth IEEE International Conference on Tools with Artificial Intelligence*, Nov 1996, pp. 467–469.

[38] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an efficient SAT solver," in *Proc. of the 38th Annual Design Automation Conference*, ser. DAC '01. New York, NY, USA: ACM, 2001, pp. 530–535. [Online]. Available: http://doi.acm.org/10.1145/378239.379017

[39] N. Sörensson and N. Een, "MiniSat v1.13-a SAT solver with conflict-clause minimization," in *Proc. of Int. Conference on Theory and Applications of Satisfiability Testing*, 01 2005.

[40] C. Archambeau, E. Peeters, F. X. Standaert, and J. J. Quisquater, "Template attacks in principal subspaces," in *Cryptographic Hardware and Embedded Systems*, 2006, pp. 1–14.

[41] H. Maghrebi, T. Portigliatti, and E. Prouff, "Breaking cryptographic implementations using deep learning techniques," in *Security, Privacy, and Applied Cryptography Engineering*, C. Carlet, M. A. Hasan, and V. Saraswat, Eds. Cham: Springer International Publishing, 2016, pp. 3–26.

[42] B. Timon, "Non-profiled deep learning-based side-channel attacks," IACR Cryptology ePrint Archive, 2018:196, 2018.

[43] E. Brier, C. Clavier, and F. Olivier, "Correlation power analysis with a leakage model," in *Cryptographic Hardware and Embedded Systems - CHES 2004*, M. Joye and J.-J. Quisquater, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 16–29.

[44] Q. Do, B. Martini, and K.-K. R. Choo, "The role of the adversary model in applied security research," *Computers & Security*, vol. 81, pp. 156–181, 2019.

[45] D. Das, A. Golder, J. Danial, S. Ghosh, A. Raychowdhury, and S. Sen, "X-DeepSCA: Cross-device deep learning side channel attack," in *Proc. of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[46] C. Liu, A. Kuehlmann, and M. Moskewicz, "Cama: a multi-valued satisfiability solver," in *ICCAD-2003. International Conference on Computer Aided Design (IEEE Cat. No.03CH37486)*, 2003, pp. 326–333.

[47] J. C. Muzio, D. M. Miller, and S. L. Hurst, *Spectral Techniques in Digital Logic*, ser. Microelectronics and Signal Processing. Orlando, FL, USA: Academic Press, 1985.