Applications of SAT Solvers to AES key Recovery from Decayed Key Schedule Images

Abdel Alim Kamal and Amr M. Youssef
Concordia Institute for Information Systems Engineering
Concordia University, Montreal, Canada
{*a_kamala,youssef*}*@ciise.concordia.ca*

***Abstract*—Cold boot attack is a side channel attack which exploits the data remanence property of random access memory (RAM) to retrieve its contents which remain readable shortly after its power has been removed. Given the nature of the cold boot attack, only a corrupted image of the memory contents will be available to the attacker. In this paper, we investigate the use of an off-the-shelf SAT solver, CryptoMinSat, to improve the key recovery of the AES-128 key schedules from its corresponding decayed memory images. By exploiting the asymmetric decay of the memory images and the redundancy of key material inherent in the AES key schedule, rectifying the faults in the corrupted memory images of the AES-128 key schedule is formulated as a Boolean satisfiability problem which can be solved efficiently for relatively very large decay factors. Our experimental results show that this approach improves upon the previously known results.**

***Keywords*-AES; Cold-boot attacks; decayed memory; SAT solvers**

I. Introduction

Cold boot attack [1] is a side channel attack that exploits the fact that data loss of a non-powered random access memory can be retarded by cooling it down. In 2002, Skorobogatov [2] performed some experiments to study the temperature dependency of data retention time in static RAM devices. The reported experimental results indicated that many chips may preserve data for relatively long periods of time at temperatures above $-20°$C which contradicted the common wisdom that was widely believed at that time. The temperature at which 80% of the data remained for one minute varied widely between devices. While some devices required cooling to at least $-50°$C, others, surprisingly, retained data for this period at room temperature. Memory retention time also varied between devices of the same type from the same manufacturer, most likely, because controlling data retention time is not a part of the chip manufacturing quality process.

Thus, one way to launch a cold boot attack is to remove the memory modules, after cooling it, from the target system and immediately plug it in another system under the adversary's control. This system is then booted to access the memory. Another possible approach to execute the attack is to cold boot the target machine by cycling its power off and then on without letting it shut down properly. Then a lightweight operating system is, instantly, booted where the contents of targeted memory are dumped to a file. Further analysis can then be performed against the information that is retrieved from memory in order to find sensitive information such as cryptographic keys or passwords.

Because of the nature of the cold boot attack, it is realistic to assume that only a corrupted image of the contents of memory will be available to the attacker, i.e., a fraction of the memory bits will be flipped. Halderman et *al.* [1] observed that, within a specific memory region, the decay is overwhelmingly asymmetric, i.e., either $0 \rightarrow 1$ or $1 \rightarrow 0$. When trying to retrieve cryptographic keys, the decay direction for a region can be determined by comparing the number of 0's and 1's since in an uncorrupted key, the expected number of 0's and 1's should approximately be equal.

Given this asymmetric decay, rectifying these faults can be achieved by further exploiting the redundancy of key material inherent in many widely used cryptographic algorithms. For example, in [3], Heninger *et al.* showed that an RSA private key with small public exponent can be efficiently recovered given a 0.27 fraction of its bits at random. In [1], Halderman *et al.* have developed a recovery algorithm for the 128-bit version of the Advanced Encryption Standard (AES-128) that recovers keys from 30% decayed AES-128 Key Schedule images in less than 20 minutes about half the time. Tsow [4] further improved upon this proof of concept and presented a heuristic algorithm that solved all cases at 50% decay and less in under half a second. At 60% decay, Tsow recovered the worst case in 35.500 seconds while solving the average case in 0.174 seconds. At the extended decay rate of 70%, recovery time averages grew to over 6 minutes with the median time at about five seconds. Nearly half of the 17.4 day run was consumed by solving the worst case of the test suite; the second slowest case was over six times faster.

It should be noted, however, that the algorithm developed by Tsow is a bit complex and was certainly uneasy to develop and fine tune. On the other hand, the relations that have to be satisfied between the different subround key bits in the AES key schedule can be easily formulated as a Boolean satisfiability (SAT) problem which lends itself

naturally to SAT solvers. In this paper, we explore the use the CryptoMiniSAT SAT solver [5] to the above problem, i.e., to recover AES-128 keys from its decayed key schedule images.

The rest of the paper is organized as follows. A very brief introduction to the Boolean satisfiability (SAT) problem is given in the next section where we also review some of the previous work related to the application of SAT solvers in cryptanalysis. The relevant details of the structure of the AES-128 key schedule are described in the section III. Our attack is then presented in section IV and the experimental results are given in section V. Finally, the conclusion is presented in section VI.

II. SAT solvers and its Applications to Cryptanalysis

The Boolean satisfiability (SAT) problem [6] is defined as follows: Given a Boolean formula, check whether an assignment of Boolean values to the propositional variables in the formula exists, such that the formula evaluates to true. If such an assignment exists, the formula is said to be satisfiable; otherwise, it is unsatisfiable. For a formula with m variables, there are 2^m possible truth assignments. The conjunctive normal form (CNF) is most the frequently used for representing Boolean formulas. In CNF, the variables of the formula appear in literals (e.g., x) or their negation (e.g., $\overline{x}$). Literals are grouped into clauses, which represent a disjunction (logical *OR*) of the literals they contain. A single literal can appear in any number of clauses. The conjunction (logical *AND*) of all clauses represents a formula. For example, the CNF formula $(x_1) \wedge (\overline{x}_2 \vee \overline{x}_3) \wedge (x_1 \vee x_3)$ contains three clauses: x_1, $\overline{x}_2 \vee \overline{x}_3$ and $x_1 \vee x_3$. Two literals in these clauses are positive (x_1, x_3) and two are negative ($\overline{x}_2$, $\overline{x}_3$). For a variable assignment to satisfy a CNF formula, it must satisfy each of its clauses. For example, if x_1 is true and x_2 is false, then all three clauses are satisfied, regardless of the value of x_3.

While the SAT problem has been shown to be NP-complete [6], efficient heuristics exist that can solve many real-life SAT formulations. Furthermore, the wide range of target applications of SAT have motivated advances in SAT solving techniques that have been incorporated into freely-available SAT software tools (e.g., [7] [8] [9] [10] [11]. Also see the international SAT Competitions web page [12].)

Given the versatility and effectiveness of SAT solving techniques, the use of SAT solvers in cryptanalysis has recently attracted the attention of many cryptanalysts. In the area of cryptanalysis of block ciphers, Courtois *et al.* [13] presented an attack on the KeeLoq block cipher. They showed that when about 2^{32} known plaintexts are available, KeeLoq is very week and for 30% of all keys, the full key can be recovered with complexity of 2^{28} KeeLoq encryptions. Erickson *et al.* [15] used the Grobner basis [14] attacks on SMS4 equation system over GF(2) and GF(2^8) and used the SAT solver over the GF(2) model. They implement their design in Grobner basis by Magma tool and in SAT solver by the MiniSAT tool. In [16], 6 rounds of DES are attacked with only single plaintext-ciphertext pair.

SAT solvers have also been applied to the cryptanalysis of stream ciphers. Eibach *et al.* [17] presented an experimental results over a slightly modified version of Trivium (Bivium) using a SAT solver, exhaustive search, a BDD based attack, a graph theoretic approach, and Grobner basis. Their results concluded that the initial state of the cipher is recovered and the using of SAT solver is faster than the other attacks. The full key of Hitage2 stream cipher is recovered in a few hours by using MiniSat 2.0 [18]. In [19], the full 48-bit key of the MiFare Crypto 1 algorithm was recovered in 200 seconds on a PC, given 1 known IV (from one single encryption).

Mironov and Zhang [20] described some initial results on using SAT solvers to automate certain components in cryptanalysis of hash functions of the MD and SHA families. They generated a full collisions for MD4 and MD5. De *et al.* [21] presented heuristics for solving inversion problems for functions that satisfy certain statistical properties similar to that of random functions. They demonstrate that this technique can be used to solve the hard case of inverting a popular secure hash function and inverted MD4 up to 2 rounds and 7 steps in less than 8 hours.

In this work, we used the CryptoMiniSat [5], which is an extension of MinSat (a state-of-the-art DPLL-based [22] SAT solver), refined to understand the XOR operation, which is common in cryptography, besides functions in CNF that is native to many SAT solvers.

III. Structure of the AES-128 key schedule

In this section, we briefly review the relevant details of the AES-128 key schedule [23] [24] . Bytes of initial key are denoted by $K^0_{i,j}$, where $0 \leq i,j \leq 3$ stand for the row index and column index, respectively, in the standard AES state matrix representation.

These 16 intial key bytes are bijectively mapped to 10 additional round-keys denoted by $K^{r+1}_{i,j}$, where $0 \leq r \leq 9$ stands for the number of the subkeys (rounds). The r^{th} key schedule round consists of the following transformations

$$\begin{aligned} K^{r+1}_{0,0} &\leftarrow S(K^r_{1,3}) \oplus K^r_{0,0} \oplus Rcon(r+1) \\ K^{r+1}_{i,0} &\leftarrow S(K^r_{(i+1)mod4,3}) \oplus K^r_{i,0},\ 1 \leq i \leq 3 \\ K^{r+1}_{i,j} &\leftarrow K^{r+1}_{i,j-1} \oplus K^r_{i,j},\ 0 \leq i \leq 3, 1 \leq j \leq 3, \end{aligned} \tag{1}$$

where $Rcon(\cdot)$ is a round-dependent constant and $S(\cdot)$ denotes the s-box (SubBytes) operation which is performed, on each byte of the state, by first taking the multiplicative inverse in $GF(2^8)$ using the irreducible polynomial $x^8+x^4+x^3+x+1$ and then applying an affine transformation over $GF(2)$.

Similar to any Boolean function, each one of the eight coordinate of the s-box, $S_l, l = 1 \cdots 8$, has a unique representation as a polynomial over $GF(2)$, called the algebraic normal form (ANF) which is obtained by summing up distinct products terms of $x_1, x_2, \ldots, x_n$, and can be written as

$$S_l(x_1, \ldots, x_n) = a_0 \bigoplus_{i=1}^{n} a_i x_i \bigoplus_{1 \le i < j \le n} a_{ij} x_i x_j \bigoplus \cdots \bigoplus a_{123\ldots n} x_1 x_2 \ldots x_n,$$

where $a_0, a_i, \ldots, a_{123\ldots n} \in GF(2)$.

IV. Formulating the AES key schedule as a SAT problem

The AES key-schedule, described in the previous section, is the primary source of redundancy utilized to rectify the faults in the corrupted memory images of the AES-128 key schedule. The conversion from the key schedule to the Boolean SAT problem proceeds as follows.

First, the system of equations of the AES-128 key schedule are generated according to the pseudocode in (1). In each round, the s-box equations in lines 2,3 of (1) are represented in its ANF (e.g., see Appendix A for the ANF of the first coordinate function of the s-box). Then, the terms of quadratic and higher degree are handled by noting that (for example) the logical expression

$$(x_1 \vee \overline{T})(x_2 \vee \overline{T})(x_3 \vee \overline{T})(x_4 \vee \overline{T})(T \vee \overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3 \vee \overline{x}_4) \quad (2)$$

is tautologically equivalent to $T \Leftrightarrow (x_1 \wedge x_2 \wedge x_3 \wedge x_4)$, or the $GF(2)$ equation $T = x_1 x_2 x_3 x_4$. Similar expressions exist for higher order terms.

Thus, the system of equations obtained in this step can be linearlized by introducing new variables as illustrated by the following toy example.

Example 1: Suppose we would like to find the Boolean variable assignment that satisfies the following formula

$$x_0 \oplus x_1 x_2 \oplus x_0 x_1 x_2 = 0$$

Using the approach illustrated in (2), we introduce two linearization variables, $T_0 = x_1 x_2$ and $T_1 = x_0 x_1 x_2$. Thus we have

$$\begin{array}{l} x_0 \oplus T_0 \oplus T_1 = 0, \\ (\overline{T}_0 \vee x_1) \wedge (\overline{T}_0 \vee x_2) \wedge (T_0 \vee \overline{x}_1 \vee \overline{x}_2) = 1, \\ (\overline{T}_1 \vee x_0) \wedge (\overline{T}_1 \vee x_1) \wedge (\overline{T}_1 \vee x_2) \wedge \\ (T_1 \vee \overline{x}_0 \vee \overline{x}_1 \vee \overline{x}_2) = 1. \end{array} \quad (3)$$

Since the CryptoMinSAT expects only positive clauses and the CNF form does not have any constants, we need to overcome the problem that the first line in (3) corresponds to a negative, i.e., false, clause. Adding the clause consisting of a dummy variable, d, or equivalently $(d \wedge d \cdots \wedge d)$ would require the variable d to be true in any satisfying solution, since all clauses must be true in any satisfying solution. In other words, the variable d will serve the place of the constant 1.

Therefore, the above formula can be reexpressed as

$$\begin{array}{l} d = 1, \\ x_0 \oplus T_0 \oplus T_1 \oplus d = 1, \\ (\overline{T}_0 \vee x_1) \wedge (\overline{T}_0 \vee x_2) \wedge (T_0 \vee \overline{x}_1 \vee \overline{x}_2) = 1, \\ (\overline{T}_1 \vee x_0) \wedge (\overline{T}_1 \vee x_1) \wedge (\overline{T}_1 \vee x_2) \wedge \\ (T_1 \vee \overline{x}_0 \vee \overline{x}_1 \vee \overline{x}_2) = 1. \end{array}$$

Figure 1 shows the CryptoMiniSat input corresponding to the above example.

```
c Lines starting with 'c' are comments
c The first line in the SAT file is in the form:
c 'p cnf # variables # clause'
c Each line should end with '0'
c Lines starting with 'x' denote XOR equations
c True variables are denoted by numbers
c False variables are denoted by negating these numbers
c In this example, d → 1, x_0 → 2 (consequently x̄_0 → −2)
c x_1 → 3, x_2 → 4, T_0 → 5, T_1 → 6
p cnf 6 9
1 0
x 2 5 6 1 0
-5 3 0
-5 4 0
5 -3 -4 0
-6 2 0
-6 3 0
-6 4 0
6 -2 -3 -4 0
```

Figure 1. CryptoMinSAT input corresponding to Example 1

In our AES-128 key schedule system, each s-box can be represented by 8 XOR equations corresponding to its 8 Boolean coordinates; and 246 CNF equations corresponding to 246 linearization variables. The total number of clauses corresponding to these CNF equations is equal to 1254. Since, each round in the AES-128 key schedule involves four s-box look-up operations and 96 linear XOR equations (line 4 in (1)), then the total number of clauses (including XOR clauses) in each round is equal to $4 \times (1254 + 8) + 96 = 5144$. Thus, for the complete 10 rounds key schedule, we have 10×5144 clauses $+1$ dummy variable to present the constant 1.

V. Experimental results

Similar to the previous work in [1] [4], throughout our experimental results, we assume an asymmetric decay model

Table I
RUN-TIME STATISTICS FOR DECAY FACTORS 30%, 40%, 50%, 60%, AND 70%.

β		30%	40%	50%	60%	70%
This work	Min	0.046	0.046	0.062	0.062	0.078
	Max	0.593	0.140	0.187	0.593	207.171
	Avg.	0.064	0.066	0.074	0.102	1.233
	St.Dev	0.009	0.007	0.008	0.028	4.899
	Med.	0.062	0.062	0.078	0.093	0.359
[4]	Min	0.000[a]	0.000	0.000	0.000	0.000
	Max	0.015	0.015	0.078	2.094	737,266.687
	Avg.	0.009	0.009	0.014	0.174	300.897
	St.Dev	0.007	0.008	0.015	0.772	10,677.913
	Med.	0.015	0.015	0.015	0.031	4.938

a. The value 0.000 means that it less than 1/64

where bits overwhelmingly decay to their ground state rather than their charged state. Using this model, only the bits that remain in their charged state will be useful to the cryptanalyst since one cannot be sure about the original values of the 0 bits, i.e., whether they were originally 0's or decayed 1's. Let β denote the fraction of decayed bits. If the percentage of 0's and 1's in the original key schedule bits is p_z and $1 - p_z$, respectively, then the fraction, f, of key bits that can be assumed to be known by examining the decayed memory of the AES key schedule is given by

$$f = 1 - (p_z + \beta \times (1 - p_z)) = (1 - p_z) \times (1 - \beta).$$

Since in an uncorrupted AES key schedule key, we expect the number of 0's and 1's to be approximately equal, i.e., $p_z \approx 1/2$, then we have $f \approx (1 - \beta)/2$.

Table I shows a comparison between our work and the results reported in [4] which recover the AES-128 key schedule from its decayed memory images for a decay factor up to 70%. In this table, for our work, all statistics were generated using 10,000 runs for each decay factor.

It should be noted that the performance in [4] was evaluated on Dell Precision Workstation 7400 running a 3.4 Ghz quad-core Xeon processor with 4GB of RAM. The performance of our approach was evaluated on a slightly less powerful machine: Dell Precision 370 Workstation running a 3.0 GHz Intel Pentium 4 CPU with 1 GB of RAM.

Table II
RUN-TIME STATISTICS FOR DECAY FACTORS 72%, 74%, 76%, 78%, AND 80%.

β	72%	74%	76%	78%	80%
Min	0.078	0.109	0.156	0.625	38.65[b]
Max	109794	126772	84819	19987	5523.8[b]
Avg.	22.271	62.404	799.327	6958.473	1901.95[b]
St.Dev	1155.83	1373.14	5423.73	25880.70	2119.58[b]
Med.	0.812	2.656	14.578	173.508	685.046[b]
# Tests	10000	10000	1000	100	7

b. These statistics excludes the 8^{th} case where the search did not terminate for 10 days

While the algorithm in [4] runs slightly faster for small values of $\beta \leq 50\%$, it is clear that our proposed SAT approach outperforms the algorithm in [4] for large values of β. In [4] with decay factor 70%, the recovery time for the worst case was more than 8.5 days. The average time was 5 minutes and median time was about five seconds. In our work, the worst case for the recovery time was obtained in less than 3.5 minutes and 7968 cases were recovered in less than one second. The average and median recovery times are are 1.2, 0.36 seconds respectively. It should also be noted that for small values of β, the difference in the run time statistics between the two approaches is practically not very significant because the run time is usually very short for such small values of β.

Table II shows the time statistics of our work corresponding to decay factors between $72\% - 80\%$. For such a large value of decay factors, the median is a better indication to the performance of the algorithm than the average since some few cases may take a relatively very long time while the majority of the cases take a short time. At 72% and 74% the results are promising, the 10,000 cases have been solved in an average of 22.3 and 62.4 seconds, respectively. At 72% decay factor, 92% of the cases have been recovered in less than 10 seconds, while similar percentage has been recovered in less than one minute with 74% decay factor. Due to the extended times for key recovery with decay factors 76%, 78%, and 80%, less cases have been examined. For $\beta = 80\%$, the CryptoMiniSat search did not terminate (at the time of submission, still running for 10 days) in the 8^{th} case. Furthermore, in one of other successfully terminated 7 cases, the key returned by the SAT solver was different from the original key. The original key could have been easily found by re-running the SAT solver again after adding some clauses to exclude the found key.

VI. CONCLUSIONS

In this work, we modelled the problem of key recovery of the AES-128 key schedules from its corresponding decayed memory images as a Boolean SAT problem and solved it using the CryptoMiniSat solver. Our experimental results confirm the versatility of our proposed approach which allows us to efficiently recover the AES-128 key schedules for large decay factors. The method presented in this work can be extended in a straightforward way to AES-192, AES-256 and other ciphers with key schedules that can be presented as a set of Boolean equations and, hence, lend themselves naturally to SAT solvers.

REFERENCES

[1] 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*, in Proc. 17th USENIX Security Symposium (Sec 08), San Jose, CA, July 2008.

[2] S. Skorobogatov, *Low temperature data remanence in static RAM*, University of Cambridge, Computer Laboratory, June 2002. Available at: http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-536.html, accessed 10 Feb. 2010.

[3] N. Heninger, H. Shachan, *Reconstructing RSA Private keys from Random key Bits,* In proc. of CRYPTO, LNCS 5677, pp. 1-17, Springer, 2009.

[4] A. Tsow, *An Improved Recovery Algorithm for Decayed AES Key Schedule Images,* In Proc. of Selected Areas in Cryptography, SAC 2009,pp. 215-230, LNCS, Springer-Verlag, 2009

[5] M. Soos, K. Nohl, and C. Castelluccia, *Extending SAT Solvers to Cryptographic Problems,* In proc. of SAT, LNCS 5584, pp. 244-257, Springer, 2009.

[6] S. Cook, *The complexity of theorem proving procedures,* in proc. of 3^{rd} Annual ACM Symposium on Theory of Computing, pp. 151-158, 1971.

[7] N. Een and N. Sörensson, *An extensible SAT-solver*, in Proc. of SAT 2003, Volume 2919, LNCS, Springer, pp. 502-518, 2004.

[8] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, S. Malik, *Chaff: Engineering an Efficient SAT Solver*, Proc. of the 39th. DAC, Las Vegas USA, 2001.

[9] E. Goldberg, Y. Novikov, BerkMin: a fast and robust SAT-solver, In Design, Automation and Test in Europe (DATE02), pp.142-149, 2002.

[10] SAT4J, a SATisfiability library for java. http://www.sat4j.org, accessed 15 Jan. 2010.

[11] M. Heule and Hans van Maaren, $March_dl$*: Adding Adaptive Heuristics and a New Branching Strategy*, Journal on Satisfiability, Boolean Modeling and Computation 2:47-59, 2006.

[12] The international SAT Competitions web page. Avilavle at http://www.satcompetition.org/

[13] N. Courtois, G. Bard, W. David *Algebraic and slide attacks on KeeLoq,* In proc. of FSE, LNCS 5086, pp. 97115, Springer, 2008.

[14] B. Buchberger, *Grobner bases: An Algorithmic method in Polynomial Ideal theory, in Multidimensional Systems Theory,* N.K. Bose, cd., D. Reidel Publishing Co., 1985.

[15] J. Erickson, J. Ding, C. Christensen *Algebraic cryptanalysis of SMS4: Grobner basis attack and SAT attack compared,* In proc. of ICISC, 2009.

[16] N. Courtois, G. Bard *Algebraic Cryptanalysis of the Data Encryption Standard,* in proc. of Cryptography and Coding, LNCS 4887, pp. 152-169, Springer, 2007.

[17] T. Eibach, E. Pliz, G. Volkel, *Attacking Bivium Using SAT solvers,* In proc. of SAT, LNCS 4996, pp. 63-76, Springer, 2008.

[18] N. Courtois, S. O'Neil, J. Quisquater *Practical Algebraic Attacks on Hitage2 Stream Cipher,* In proc. of ISC, LNCS 5735, pp. 167176, Springer, 2009.

[19] N. Courtois, K. Nohl, S. O'Neil *Algebraic Attacks on the Crypto-1 Stream Cipher in MiFare Classic and Oyster Cards,* Cryptology ePrint archive, report 166, 2009.

[20] I. Mironov and L. Zhang, *Applications of SAT Solvers to Cryptanalysis of Hash Functions,* in proc. of International Symposium on the Theory and Applications of Satisfiability and Testing (SAT), LNCS 4121, pp. 102115, Springer, 2006.

[21] D. De, A. Kumarasubramanian, R. Venkatesan, *Inversion Attacks on Secure Hash Functions Using SAT Solvers,* in proc. of International Symposium on the Theory and Applications of Satisfiability and Testing (SAT), LNCS 4501, pp. 377382, Springer, 2007.

[22] M. Davis, H. Putnam, *A Computing Procedure for Quantification Theory*, Journal of the ACM Vol. 7, Issue 3, pp. 201-215, 1960.

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

[24] Federal Information Processing Standards Publication (FIPS 197), *Advanced Encryption Standard (AES)* , Nov. 26, 2001.

APPENDIX

$$
\begin{aligned}
S_1 = {} & x_1 \oplus x_3 \oplus x_4 \oplus x_6 \oplus x_1x_3 \oplus x_1x_7 \oplus x_1x_8 \oplus x_2x_4 \oplus \\
& x_2x_6 \oplus x_2x_8 \oplus x_3x_5 \oplus x_4x_6 \oplus x_6x_7 \oplus x_6x_8 \oplus \\
& x_1x_2x_4 \oplus x_1x_2x_6 \oplus x_1x_2x_8 \oplus x_1x_3x_4 \oplus x_1x_3x_5 \oplus \\
& x_1x_4x_7 \oplus x_1x_5x_8 \oplus x_1x_6x_8 \oplus x_2x_3x_4 \oplus x_2x_3x_6 \oplus \\
& x_2x_3x_8 \oplus x_2x_4x_6 \oplus x_2x_5x_7 \oplus x_2x_5x_8 \oplus x_2x_6x_7 \oplus \\
& x_2x_7x_8 \oplus x_3x_4x_5 \oplus x_3x_4x_8 \oplus x_3x_5x_6 \oplus x_3x_5x_7 \oplus \\
& x_3x_5x_8 \oplus x_3x_6x_8 \oplus x_3x_7x_8 \oplus x_4x_7x_8 \oplus x_5x_6x_7 \oplus \\
& x_5x_6x_8 \oplus x_1x_2x_3x_4 \oplus x_1x_2x_3x_6 \oplus x_1x_2x_3x_7 \oplus \\
& x_1x_2x_4x_6 \oplus x_1x_2x_4x_7 \oplus x_1x_2x_4x_8 \oplus x_1x_2x_5x_7 \oplus \\
& x_1x_2x_5x_8 \oplus x_1x_3x_4x_7 \oplus x_1x_3x_5x_8 \oplus x_1x_4x_5x_6 \oplus \\
& x_1x_4x_5x_7 \oplus x_1x_4x_5x_8 \oplus x_1x_4x_6x_8 \oplus x_1x_4x_7x_8 \oplus \\
& x_1x_5x_6x_7 \oplus x_1x_5x_6x_8 \oplus x_1x_6x_7x_8 \oplus x_2x_3x_4x_5 \oplus \\
& x_2x_3x_5x_8 \oplus x_2x_4x_5x_8 \oplus x_2x_4x_6x_7 \oplus x_2x_4x_6x_8 \oplus \\
& x_2x_5x_6x_8 \oplus x_2x_5x_7x_8 \oplus x_3x_4x_6x_7 \oplus x_3x_5x_6x_7 \oplus \\
& x_3x_5x_6x_8 \oplus x_3x_6x_7x_8 \oplus x_4x_5x_6x_7 \oplus x_4x_5x_7x_8 \oplus \\
& x_4x_6x_7x_8 \oplus x_5x_6x_7x_8 \oplus x_1x_2x_3x_4x_5 \oplus x_1x_2x_3x_4x_7 \oplus \\
& x_1x_2x_3x_5x_7 \oplus x_1x_2x_3x_5x_8 \oplus x_1x_2x_3x_6x_7 \oplus \\
& x_1x_2x_3x_7x_8 \oplus x_1x_2x_4x_5x_8 \oplus x_1x_2x_4x_6x_7 \oplus \\
& x_1x_2x_6x_7x_8 \oplus x_1x_3x_4x_7x_8 \oplus x_1x_3x_6x_7x_8 \oplus \\
& x_1x_4x_5x_6x_8 \oplus x_1x_4x_5x_7x_8 \oplus x_2x_3x_4x_5x_6 \oplus \\
& x_2x_3x_4x_5x_8 \oplus x_2x_3x_4x_6x_7 \oplus x_2x_3x_4x_7x_8 \oplus \\
& x_2x_3x_5x_6x_7 \oplus x_2x_4x_5x_6x_8 \oplus x_2x_4x_6x_7x_8 \oplus \\
& x_2x_5x_6x_7x_8 \oplus x_3x_4x_5x_6x_8 \oplus x_3x_4x_5x_7x_8 \oplus \\
& x_4x_5x_6x_7x_8 \oplus x_1x_2x_3x_4x_5x_7 \oplus x_1x_2x_3x_4x_6x_8 \oplus \\
& x_1x_2x_3x_5x_6x_8 \oplus x_1x_2x_3x_5x_7x_8 \oplus x_1x_2x_4x_5x_6x_8 \oplus \\
& x_1x_2x_4x_5x_7x_8 \oplus x_1x_2x_5x_6x_7x_8 \oplus x_1x_3x_4x_5x_6x_8 \oplus \\
& x_1x_3x_4x_5x_7x_8 \oplus x_1x_3x_4x_6x_7x_8 \oplus x_2x_3x_4x_5x_6x_8 \oplus \\
& x_1x_2x_3x_4x_5x_6x_8 \oplus x_1x_2x_3x_4x_5x_7x_8
\end{aligned}
$$