Algebraic Side-Channel Attack on Twofish

Chujiao Ma*, John Chandy, and Zhijie Shi
University of Connecticut, Storrs, Connecticut 06269, USA
{chujiao.ma, john.chandy, zhijie.shi}@uconn.edu

Abstract

While algebraic side-channel attack (ASCA) has been successful in breaking simple cryptographic algorithms, it has never been done on larger or more complex algorithms such as Twofish. Compared to other algorithms that ASCA has been used on, Twofish is more difficult to attack due to the key-dependent S-boxes as well as the complex key scheduling. In this paper, we propose the first algebraic side-channel attack on Twofish, and examine the importance of side-channel information in getting past the key-dependent S-boxes and the complex key scheduling. The cryptographic algorithm and side-channel information are both expressed as boolean equations and a SAT solver is used to recover the key. While algebraic attack by itself is not sufficient to break the algorithm, with the help of side-channel information such as Hamming weights, we are able to correctly solve for 96 bits of the 128 bits key in under 2 hours with known plaintext/ciphertext.

Keywords: algebraic side-channel attack, Twofish, cryptography, block cipher, SAT solver, conjunctive normal form

1 Introduction

Twofish is a symmetric key block cipher and one of the five finalists of the Advanced Encryption Standard contest. It is generally recognized as a secure block cipher due to its unique implementation and is used in many areas from file encryption to password management. Usually, a block cipher would divide the plaintext into blocks, then perform rounds of operations and combine them with the key to produce the ciphertext. The operations in each round typically consist of XORing the input with the key, S-boxes, shifts and multiplications. For Twofish, however, each block is further divided into four, each of which is put through different operations in each round. Twofish also uses key-dependent S-boxes and has a relatively complex key scheduling. Because the S-boxes are key-dependent, Twofish is thus immune to many attacks based on traditional S-box statistics. The round keys used in the cryptographic algorithms are generated in key scheduling from the master key. Not only does the key scheduling of Twofish uses complex operations, each round key is calculated with a key-dependent S-box as well [18]. While there have been some promising attacks on Twofish [13, 6], they are done on reduced-round versions and not the full version. Differential fault analysis has been a little more successful on Twofish. By focusing on the S-box keys, the attack is able to acquire the complete key under 8 hours [1]. However, fault injection is an active attack that interferes with the cryptographic device. It is, thus, detectable and can be countered with error-correcting countermeasures.

This paper examines whether Twofish can be broken via algebraic side-channel attack (ASCA), a passive attack where the cryptographic algorithm and side-channel information are expressed as a set of boolean equations. The equations are then put through a solver to find the secret key. The practicality and success of an algebraic attack depend on the amount of information that can be expressed as equations and, moreover, the information must be correct. When there is insufficient information, the solver may

Journal of Internet Services and Information Security (JISIS), volume: 7, number: 2 (May 2017), pp. 32-43
*Corresponding author: Department of Computer Science and Engineering, 371 Fairfield Way, Unit 4155, University of Connecticut, Storrs, CT 06269-4155, USA.

output no solution or multiple solutions. If the information has a margin of error, then the results may not be accurate. However, it is possible to increase the success rate of finding the key by combining different attacks, or in this case, combining algebraic analysis with side-channel information. Side-channel attacks are based on the idea that seemingly harmless leakage from the cryptosystem, such as power consumption or electromagnetic waves, can be exploited to break the cryptographic algorithm and find the key. However, this attack is dependent on the accuracy of the data, knowledge of the plaintext or ciphertext and only targets one key byte at a time. By combining both attacks, algebraic side-channel attack can bypass some of those problems [15]. ASCA is different from side-channel attacks in that it potentially exploits leakages of all rounds instead of just the first or the last round, and the solver will output the complete key at once. By incorporating algebraic analysis with side-channel attack, it can also succeed in unknown plaintext/ciphertext scenario and requires less leakage information.

The remainder of the paper is organized as follows: section 2 starts with an overview of the Twofish algorithm and its unique characteristics. Section 3 examines how the algebraic side-channel attack works, while the attack is implemented in section 4. The results of the attack is analyzed in section 5 and the paper concludes in section 6.

2 TwoFish Overview

Twofish is a symmetric key block cipher with key sizes of 128 bits, 192 bits or 256 bits. For our implementation, Twofish-128 is used. The following notations are used to describe the algorithms:

- P for plaintext
- C for ciphertext
- K for key
- R for round
- F and g represent the functions in each round

There are three sets of keys used in Twofish, S_0 and S_1 for the key-dependent S-boxes in each round of Twofish, M_e and M_o for the key-dependent S-boxes in key scheduling, and 40 round keys (K_0, K_1...K_{39}). As for the actual Twofish algorithm, it first splits the block of 128 bits plaintext into four 32-bit words. The four words are then XORed with the subkey K_0, K_1, K_2 and K_3 during input whitening. The result is then put through 16 Feistel rounds, where two keys are used in each round from (K_8, K_9) to (K_{38}, K_{39}), then XORed with another set of subkeys in the output whitening phase (K_4, K_5, K_6 and K_7) to produce the final ciphertext.

2.1 Round Operations

In each round, the block of 128 bits is first divided into four words (32 bits). The first two words of round R_i are put through the F function, the outputs are XORed with the last two words from R_i to become the first two words of the next round, R_{i+1}. The first two words of R_i become the last two words of the next round R_{i+1}. The round function uses a Feistel network structure, a general method of transforming any function into a permutation. The operations in the F function of Twofish consist of:

- S-box substitution of bytes
- MDS matrix mixing of bytes

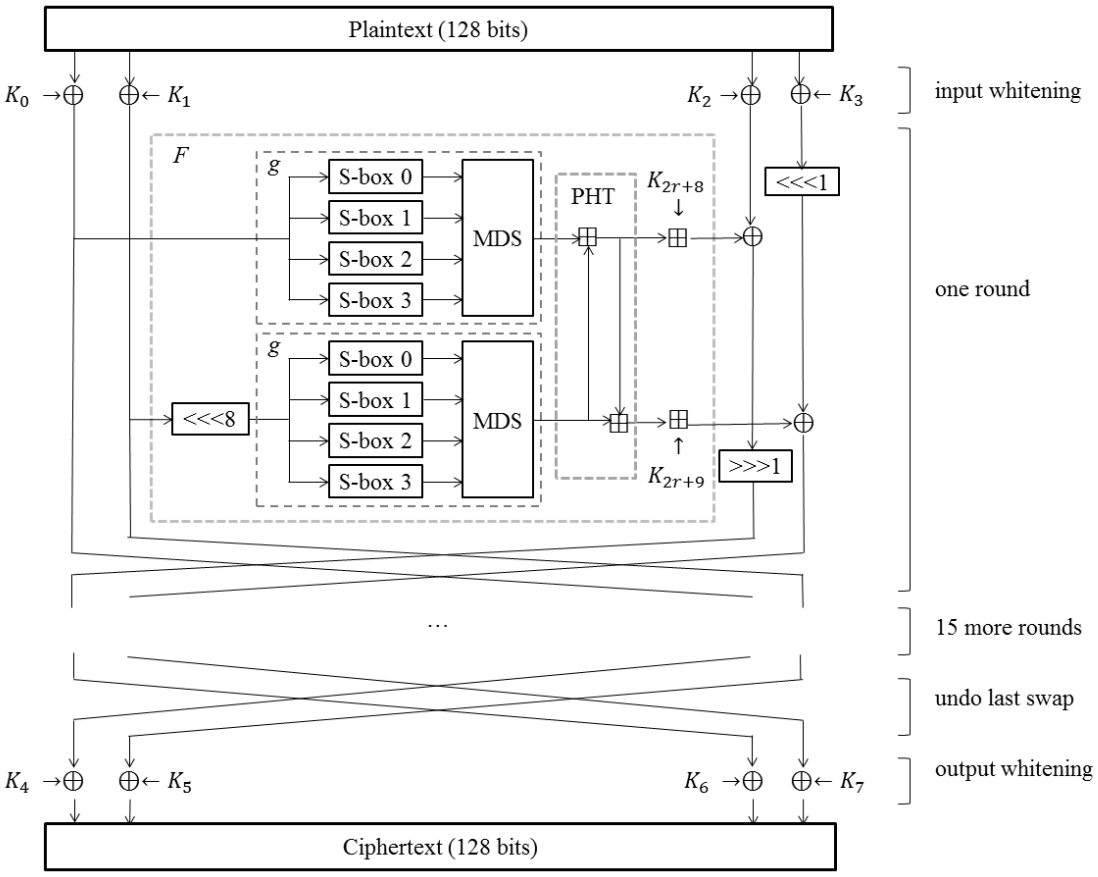

Figure 1: Twofish

- PHT mixing of words
- Sub-key modular add of words

Equation 1 illustrates the operations of the F function where the inputs are R_0 and R_1, and the outputs are F_0 and F_1. The round number r is another input of F and is used to calculate the round key for that round.

$$\begin{aligned} T_0 &= g(R_0) \\ T_1 &= g(ROL(R_1, 8)) \\ F_0 &= (T_0 + T_1 + K_{2r+8}) mod 2^{32} \\ F_1 &= (T_0 + 2T_1 + K_{2r+9}) mod 2^{32} \end{aligned} \tag{1}$$

The g function in the F function consists of the key-dependent S-box and matrix multiplication. Because the S-boxes of most algorithms are not key-dependent, they are commonly expressed as lookup tables. However, the S-box step of Twofish in Figure 1 actually consists of a series of operations. The input word of the S-box is first divided into four bytes. Each byte, x, is inputted into the permutation functions as detailed in equation 2, where a and b are the first and last 4 bits of the input, t_0, t_1, t_2 and t_3 are 4-bit lookup tables and y is the output. The output of the permutation function is XORed with the S-box key S_0, the process is repeated again with S-box key S_1, then it goes through a last round of

permutation.

$$
\begin{aligned}
&a_0, b_0 = \lfloor x/16 \rfloor, x mod 16 \\
&a_1 = a_0 \oplus b_0 \\
&b_1 = a_0 \oplus ROR_4(b_0, 1) \oplus 8a_0 mod 16 \\
&a_2, b_2 = t_0\lfloor a_1 \rfloor, t_1\lfloor b_1 \rfloor \\
&a_3 = a_2 \oplus b_2 \\
&b_3 = a_2 \oplus ROR_4(b_2, 1) \oplus 8a_2 mod 16 \\
&a_4, b_4 = t_2\lfloor a_3 \rfloor, t_3\lfloor b_3 \rfloor \\
&y = 16b_4 + a_4
\end{aligned}
\tag{2}
$$

After the four bytes of the state go through the S-box function, they are then combined and multiplied with a matrix of constants for the MDS step. This completes the g function part of the F function for each round [17].

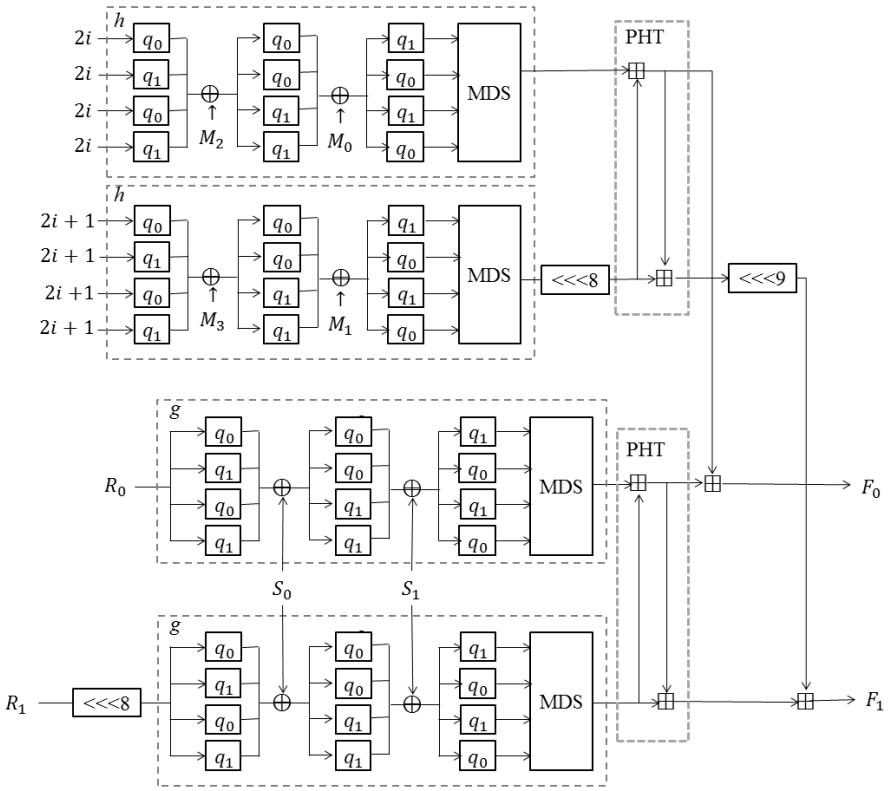

Figure 2: The g function for each round of Twofish, and h function for the key scheduling.

2.2 Key Scheduling

Key scheduling produces the two sets of keys used in the Twofish algorithm: the keys for the S-boxes, denoted as S_0, S_1, and 40 keys for the input/output whitening and each round. To calculate the S-box keys, the 128-bit master key is first separated into four words, M_0, M_1, M_2 and M_3. The first two words are multiplied with a matrix of constants, RS, to produce S_0 and the last two words produce S_1 as shown in equation 3.

$$\begin{pmatrix} s_{i,0} \\ s_{i,1} \\ s_{i,2} \\ s_{i,3} \end{pmatrix} = RS \cdot \begin{pmatrix} m_{8i} \\ m_{8i+1} \\ m_{8i+2} \\ m_{8i+3} \\ m_{8i+4} \\ m_{8i+5} \\ m_{8i+6} \\ m_{8i+7} \end{pmatrix} \tag{3}$$

To produce the round keys K_0 to K_{39}, the key scheduling uses h function, which is the same as the g function for each round of Twofish, as illustrated in Figure 2. The only difference being that key scheduling uses M_0, M_1, M_2 and M_3 from the master key for the S-boxes and the inputs to the h function are $2i$ and $2i+1$ where the value of i goes from 0 to 19. This produces 40 round keys, where K_0 to K_7 are used for input/output whitening and K_8 to K_{39} are used during each round of Twofish as illustrated in Figure 1 [16].

3 Attack Model

The attack used in this paper is algebraic side-channel attack. Algebraic analysis is a mathematical attack where the attacker expresses the cryptographic algorithm as a system of equations, then puts it through a solver, along with known variables such as the plaintext or ciphertext, to solve for the key. However, depending on the complexity of the algorithm, information from only the algorithm may not be enough to solve for the key. Side-channel attacks (SCA) use information leaked during the encryption or decryption of the algorithm and the knowledge of the algorithm to guess the key using brute force attack and statistical analysis. One of the most popular SCAs is Differential Power Analysis [11]. It is based on the idea that the power consumption and the intermediate variables of the algorithm are directly related. For each key byte, we categorize the power consumption data based on Hamming weights calculated from all possible key values. The key guess that produces the greatest difference in the categorized power consumption is the correct key. The more advanced Correlation Power Analysis (CPA) uses the same method, except we are looking for best correlation instead of maximum difference.

Algebraic Side-Channel Attack combines both algebraic and side-channel attacks. The Hamming weights information acquired from SCA and equations of the cryptographic algorithm are both expressed as a system of equations and put through a solver [15]. While ASCA has been done successfully on smaller algorithms with 64-bit keys, it has never been done on larger or more complex algorithm with 128-bit keys like Twofish. Compare to other algorithms that ASCA has been used on, Twofish is also more difficult to attack due to the key-dependent S-boxes as well as the complex key scheduling.

4 Technical Approach

To launch the attack, we first model the algorithm and the side-channel derived Hamming weights as a set of conjunctive normal form (CNF) equations. The equations are then put through the CryptoMiniSAT

SAT solver to solve for the key.

4.1 Twofish CNF Form

CryptoMiniSAT and most SAT solvers require the input to be in CNF, which is a conjunction (AND) of clauses, with each clause being a disjunction (OR) of literals. The literals are boolean variables, with positive variable representing 1 and negative variable representing 0. The version of CryptoMiniSAT used also supports XOR operations.

The CNF equation literals represent the input and output bits as well as all intermediate variables in Twofish. The round and F function of Twofish are based on the following key operations:

- XOR for byte and word
- SHIFT for byte and word
- ADD for word
- MULT used in RS and MDS matrix multiplication
- SBOX implemented as byte-wise lookup tables

While XOR and SHIFT are not difficult to represent in CNF, converting ADD, MULT, and SBOX to binary equations in CNF proves to be quite challenging. ADD requires extra intermediate variables to function as carryover during the operation.

Table 1: MDS

Y	01	02	EF	5B
y_0	x_0	x_1	$x_0+x_6+x_7$	x_0+x_6
y_1	x_1	x_0+x_2	$x_0+x_1+x_7$	x_1+x_7
y_2	x_2	x_0+x_3	$x_0+x_1+x_2+x_6+x_7$	$x_0+x_2+x_6$
y_3	x_3	x_4	$x_1+x_2+x_3+x_6$	$x_1+x_3+x_6+x_7$
y_4	x_4	x_0+x_5	$x_2+x_3+x_4+x_7$	$x_2+x_4+x_7$
y_5	x_5	x_6	$x_3+x_4+x_5+x_6+x_7$	$x_3+x_5+x_6$
y_6	x_6	x_7	$x_4+x_5+x_6+x_7$	$x_4+x_6+x_7$
y_7	x_7	x_0	$x_5+x_6+x_7$	x_5+x_7

MULT is byte-wise multiplication and used during matrix multiplication for the S-box key and Maximum Distance Separable (MDS) step of the round function. In order to multiply on a bit level, the two bytes to be multiplied are expressed as polynomial equations in Galois Field where each bit is represented by a variable. The polynomial equations are multiplied and reduced, after which the coefficients of the binary variables become the equation for the resulting byte of the multiplication [2]. Table 1 shows the CNF equations for the MDS matrix multiplication when the input byte, represented by bit variables x_0 to x_7, is multiplied by MDS in equation 4 to produce the output byte, represented by bit variables y_0 to y_7. Once the equation set for byte multiplication is done, the matrix multiplication step of Twofish is just

a matter of looping through the matrix, doing byte multiplication and adding up the appropriate bytes.

$$MDS = \begin{pmatrix} 01 & EF & 5B & 5B \\ 5B & EF & EF & 01 \\ EF & 5B & 01 & EF \\ EF & 01 & EF & 5B \end{pmatrix} \tag{4}$$

The equations are derived in a similar manner for the multiplication to get the S-box key.

The SBOX step of Twofish is the most complicated component to model because it is key-dependent. Instead of just a lookup table used by most other algorithms, Twofish's SBOX consists of XORing the S-box key and the permutation functions q_0 and q_1. Both permutation functions consist of the same operations, shown in Figure 2, but use four different sets of S-boxes. In total, there are eight S-boxes, which can be represented as lookup table consists of 4-bits. The equations of the lookup table are produced using the S-box feature in SageMath, a mathematical software. Each bit of the output is an equation of monomials that are input bits of the lookup table. The 4-bit lookup table of t_0 in the function q_0 produces equation 5, where $x0$ to $x3$ are the 4-bit variables of the input and $y0$ to $y3$ are the 4-bit variables of the output:

$$\begin{aligned} y_0 =& 1 + x_0 + x_1 + x_2 + x_3 + x_0 * x_2 + x_1 * x_2 + x_0 * x_1 * x_2 + x_0 * x_1 * x_3 + x_1 * x_2 * x_3 \\ y_1 =& x_1 + x_2 + x_0 * x_2 * x_3 \\ y_2 =& x_1 + x_2 + x_0 * x_2 + x_0 * x_3 + x_1 * x_2 + x_2 * x_3 + x_0 * x_1 * x_2 + x_1 * x_2 * x_3 \\ y_3 =& x_2 + x_3 + x_2 * x_3 + x_0 * x_1 * x_2 + x_0 * x_1 * x_3 + x_1 * x_2 * x_3 \end{aligned} \tag{5}$$

The same method is used to produce the lookup table for t_1, t_2, and t_3.

After all the basic operations have been converted to CNF form, we can then write the complete key scheduling and the round function for the Twofish algorithm. After checking that it produces the correct ciphertext, when given a plaintext and the key, we can then add SCA information into the system and begin the attack.

4.2 Side-Channel Information

For ASCA attacks, the side-channel information used is usually deduced from template attacks. The side-channel information, or the Hamming weight, counts the number of 1's in the data set and are correlated with the power consumption for the intermediate variables. In 2009, Renauld et al. showed that, with error detection and likelihood rating, the Hamming weights of intermediate values can be obtained from power traces with an error rate of at most 1% [15]. Thus, for our attack, we assume that the Hamming weights are all correct and generated them from the encryption process. While we did not perform side-channel attack to collect the Hamming weights, there are a number of approaches that have been demonstrated that one could use to collect this information. Side-channel information can be acquired by collecting power consumption data and observing the location of the power trace from template attacks. Since the power consumption does not strictly correlate with the operation and data during every step of the algorithm, where to extract Hamming weights is also important. The following operations are cryptographic computational kernels that are commonly susceptible to power leakage [12, 5, 8]:

- Data access to/from registers.
- Rotations and shifts.
- Data-dependent offset.

- Bitwise boolean operations.

In short, side-channel information such as Hamming weights can be leaked from almost every step of the algorithm from addition to XORing the key to the S-boxes. However, it is a little different for Twofish. Unlike most algorithms where every byte of the block goes through the same set of operations at the same time, Twofish divides the block into four words, each of which goes through a different set of operations and interacts with each other before being combined again for the next round. Because of this, it would be impractical to assume we can get Hamming weights for every operation within the round. Thus, for this attack only the Hamming weights of input/output whitening and between each round are used [3].

Each block of Twofish is 128 bits, or 16 bytes. To calculate the Hamming weights, we count the number of 1s in each of the 16 bytes for each round. The Hamming weight of each byte depends on the value of the byte. To model it as CNF equations, each bit of the byte is represented by variables $x_0, x_1...x_7$. The Hamming weights are expressed as sets of equations. For each possible Hamming weight value, the equation set consists of one equation that is the XOR of all bit variables and seven equations that are the ORs of the bit variables ANDed together. Equation 6 shows the set of equations for Hamming weight of 1. This means only one bit variable of the byte has a value of 1, so the XOR of all the bit variables equal to 1 while the rest of the equation equals to 0.

$$\begin{aligned} x_0 \oplus x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_5 \oplus x_6 \oplus x_7 &= 1 \\ x_0*x_1 + x_0*x_2 + x_0*x_3 + x_0*x_4 + x_0*x_5 + x_0*x_6 + x_0*x_7 + x_1*x_2 + ... + x_6*x_7 &= 0 \\ x_0*x_1*x_2 + x_0*x_1*x_3 + x_0*x_1*x_4 + x_0*x_1*x_5 + x_0*x_1*x_6 + x_0*x_1*x_7 & \\ + x_0*x_2*x_3 + ... + x_5*x_6*x_7 &= 0 \\ ... & \\ x_0*x_1*x_2*x_3*x_4*x_5*x_6*x_7 &= 0 \end{aligned} \tag{6}$$

If the Hamming weight of the byte is 2, then the equation of the two variable monomials should equal to 1 since one of the monomials should be the AND of the two bit variables with value of 1. The rest of the equations equal to 0 and so on. For Hamming weight of 8, all bits of the byte have value of 1, so the equation of XOR should equal 0 while all other equations should equal to 1.

While the accuracy of the side-channel information depends on the noise in the data collected and may compromise the accuracy of the equations for ASCA, this problem can be reduced by using multiple deductions-based algebraic side-channel attack [21]. For this attack, multiple sets of possible solutions are produced, thus maximize the possibility of finding the correct key. The attack can be further optimized by exploiting the incomplete diffusion feature in one AES round, where it can exploit the side-channel leaks in all AES rounds using a single power trace with less time complexity [7]. The noise can also be taken into account by introducing an error variable or bit in the set of Hamming weights, such attack is known as Tolerant ASCA[14].

Once the Hamming weights are acquired for each round, they are modeled as CNF equations and put in the same set as the CNF equation of the Twofish algorithm. The set of equations is then inputted into a SAT solver to solve for the key.

4.3 CryptoMiniSAT Solver

While there are many solvers available, we choose to use CryptoMiniSAT 5.0. CryptoMiniSAT has better performance than the traditional MiniSAT solver. An ASCA attack on the algorithm PRESENT-80 only need 9 rounds of leakages to obtain the full key with unknown plaintext/ciphertext instead of the 26 rounds of previous works [10].

To check for correctness, the plaintext and key are used in the standard C implementation of Twofish to generate the ciphertext. The same set of plaintext and ciphertext are given to the solver along with the ASCA equations to solve for the key.

5 Results

The ciphertexts for 100 random input-key pairs are generated for the attack. The attack is done on a PC with an Intel Haswell x64 processor. The first attempt uses algebraic attack only with no side-channel information to solve for the 128-bit key of the full algorithm. In this case, we are unable to solve for any part of the key even when the plaintext and ciphertext are given. When we attempt to solve for the full 128 bits key of the full algorithm, the solver is unable to find a solution within 7 days. However, when 32 bits of the 128 bits key are fixed, the solver is able to correctly solve for the other 96 bits under 2 hours. Since the Twofish key is separated into four words when used in the algorithm, we decide to solve for 96, 64, 32 bits of the key and analyze the performance of the attack under two scenarios:

- Is plaintext and/or ciphertext needed for the ASCA on Twofish?
- How much Hamming weight is needed?

In all the experiments and scenarios, the partial keys being solved are correct as long as the solver is able to give an output. Thus, the analysis of the paper focuses on the efficiency of the attack.

Table 2: Solving time using all Hamming weights (in seconds).

Key solved	Plaintext/Ciphertext	Plaintext	Ciphertext
1 word	51.08	31.03	35.84
2 word	959.72	-	-
3 word	3711.74	-	-

Table 3: Solving time based on Hamming weight (HW), given plaintext/ciphertext.

Key solved	All key HW	1 key HW	All Twofish HW	1 Twofish HW	All HW
1 word	184.96	75.78	89.46	20.38	51.08
2 word	-	-	-	-	959.71
3 word	-	-	-	-	3711.74

For the first case of ASCA, we analyze the importance of known plaintext or ciphertext when solving for 1 word, 2 words or 3 words of the key. When given 96 bits of the key (3 words) and solving for 32 bits of the key (1 word), the plaintext or ciphertext only scenario are actually faster than when both plaintext and ciphertext are given. This may be due to the fact that, with more information to take into account, the solver is overconstrained and thus takes longer to find the solution. However, when we attempt to solve for more parts of the key, 2 words or 3 words, the plaintext or ciphertext only scenario is unable to solve for it within 24 hours. We are also unable to solve for any part of the key when given Hamming weights only, so plaintext and ciphertext do drastically decrease the time needed for a successful attack. The details of the results are shown in Table 2.

The second scenario considered is how much Hamming weights are needed since more Hamming weights are not necessarily better. Not only can they overconstraint the solver, and thus cause it to take longer to find a solution, inaccuracies in the Hamming weights may also cause the solver to output wrong solutions. Since the Hamming weights used are from the beginning and end of each round, there are a total of 20 Hamming weights calculated from key scheduling and 17 Hamming weights calculated from Twofish. In Table 3, the average time needed to solve partial key is shown for when using all the Hamming weights from the key, only one Hamming weight from the key, when using all the Hamming weights from the Twofish rounds, when using only one Hamming weight from the Twofish round, and when using all Hamming weights from both the key scheduling and Twofish round.

The scenarios are for when plaintext and ciphertext are given. As shown in Table 3, solving for 32 bit partial key (1 word) is much faster when only one Hamming weight is used. Overall, solving for partial key is faster when using the Hamming weight for the Twofish round instead of the key scheduling. When we attempt to solve for 64 bit and 96 bit partial key (2 word and 3 word respectively), the solver is unable to output a solution within a reasonable amount of time. From this, we can conclude that even if the key scheduling is precomputed, or even if we can only calculate one Hamming weight from the Twofish algorithm, ASCA can still solve for 32 bit partial key. While less Hamming weights may allow the solver to find a solution faster, as the amount of information that needs to be solved increases, more Hamming weights may be needed.

6 Conclusion and Future Works

For algebraic side-channel attack on Twofish, we expresses the Hamming weights and the Twofish algorithm as a system of equations. The equations are then used as input for the CrytoMiniSAT solver to solve for the key. While algebraic attack by itself is not sufficient to break the algorithm, with the help of side-channel information such as Hamming weights, we are able to correctly solve for 96-bits of the 128 bits key in under 2 hours with known plaintext/ciphertext. While given both plaintext and ciphertext as well as all the Hamming weights lead to overconstraints and cause a slower solving time when solving for 32-bits of the key, more information is needed to correctly solve for more bits of the key. For this attack, we assume the Hamming weights are known and correct, which may not always be the case due to noise. However, we are able to deduce partial key with even one Hamming weight, so the attack is achievable with real Hamming weight data even if we discard the ones with more noise. It is also possible to introduce error variables for the Hamming weight equations to account for any inaccuracy.

While ASCA is unable to solve for the full 128-bits key in less than 7 days, it is able to solve for at least 96-bits of the key in less than 2 hours. In the future, we can attempt to acquire the full 128 bits of the key by combining ASCA with other attacks such as differential attacks or fault injection. Algebraic differential analysis has been used to break reduced round DES before, where the differential analysis information comes from analyzing the different plaintext/ciphertext pairs that use the same key [4]. Differential algebraic cryptanalysis is also used to break reduced round Serpent, which is another finalist of the AES competition [9]. ASCA has also been combined with fault injection to recover the key to LED block cipher [19]. The 64-bit key is recovered within one minute on a common PC with a success rate of 79%. The ASCA fault injection attack is also successfully performed on GOST, a block cipher where the S-boxes can be made known or unknown [20].

The ASCA attack proposed in this paper uses the CNF model and a SAT solver. However, the model may be simplified when used with different solvers. SAT solvers are used in verification and have NP complexity, so the functions are limited and the large set of equations is difficult to keep track of. Other optimization techniques may prove to be more efficient when used as solvers. For example,linear programming is polynomial time soluble with continuous variables. It would also be interesting to see

what results the attack would produce with mixed integer programming, or constraint programming with arbitrary constraints.

References

[1] S. S. Ali and D. Mukhopadhyay. Differential fault analysis of twofish. In *Revised Selected Papers of the 8th International Conference on Information Security and Cryptology (Inscrypt'12), Beijing, China*, volume 7763 of *Lecture Notes in Computer Science*, pages 10–28. Springer Berlin Heidelberg, 2013.

[2] C. J. Benvenuto. Galois field in cryptography, May 2012. `https://sites.math.washington.edu/~morrow/336_12/papers/juan.pdf` [Online; Accessed on May 1, 2017].

[3] S. Chari, C. Jutla, J. R. Rao, and P. Rohatgi. A cautionary note regarding evaluation of AES candidates on smart-cards. In *Proc. of the 2nd Advanced Encryption Standard (AES) Candidate Conference, Rome, Italy*, pages 133–147. NIST, March 1999.

[4] J. charles Faugère, L. Perret, and P. jean Spaenlehauer. Algebraic-differential cryptanalysis of DES, 2006. `http://www-polsys.lip6.fr/~jcf/Papers/DESweworc.pdf` [Online; Accessed on May 1, 2017].

[5] J. Daemen and V. Rijmen. Resistance against implementation attacks: A comparative study of the AES proposals. In *Proc. of the 2nd Advanced Encryption Standard (AES) Candidate Conference*, volume 82, Rome, Italy, February 1999.

[6] N. Ferguson, J. Kelsey, B. Schneier, and D. Whiting. A Twofish retreat: Related-key attacks against reduced-round Twofish. `https://www.schneier.com/academic/archives/2000/02/a_twofish_retreat_re.html` [Online; Accessed on May 1, 2017], February 2000.

[7] S. Guo, X. Zhao, F. Zhang, T. Wang, Z. J. Shi, F.-X. Standaert, and C. Ma. Exploiting the incomplete diffusion feature: A specialized analytical side-channel attack against the AES and its application to microcontroller implementations. *IEEE Transactions on Information Forensics and Security*, 9(6):999–1014, June 2014.

[8] W. Hnath. *Differential Power Analysis Side-Channel Attacks in Cryptography*. PhD thesis, Worcester Polytechnic Institute, 2010.

[9] L. Hui, X. Wang, and M. Wang. Differential-algebraic cryptanalysis of reduced-round of serpent-256. *Science China Information Sciences*, 53(3):546–556, March 2010.

[10] W. Kehui, W. Tao, Z. Xinjie, and L. Huiying. Cryptominisat solver based algebraic side-channel attack on present. In *Proc. of the 1st International Conference on Instrumentation, Measurement, Computer, Communication and Control (IMCCC'11), Beijing, China*, pages 561–565. IEEE, October 2011.

[11] P. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In *Proc. of the 19th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO'99). San Barbara, California, USA*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, Berlin, Heidelberg, August 1999.

[12] T. Lash. A study of power analysis and the advanced encryption standard: Recommendations for designing power analysis resistant devices. MS Scholarly Paper, George Mason University, 2002. `https://pdfs.semanticscholar.org/e8ee/eca036bdbcf17339bf2ea5a292fabd1a6b9b.pdf` [Online; Accessed on May 1, 2017].

[13] S. Lucks. The saturation attack — a bait for twofish. In *Revised Papers of the 8th International Workshop on Fast Software Encryption (FSE'01), Yokohama, Japan*, volume 2355 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2002.

[14] Y. Oren and A. Wool. Tolerant algebraic side-channel analysis of AES. Cryptology ePrint Archive Report 2012/092, 2012. `https://eprint.iacr.org/2012/092.pdf` [Online; Accessed on May 1, 2017].

[15] M. Renauld and F.-X. Standaert. Algebraic side-channel attacks. In *Proc. of the 5th International Conference Information Security and Cryptology (INSCRYPT'09), Beijing, China*, volume 6151 of *Lecture Notes in Computer Science*, pages 393–410. Springer, Berlin, Heidelberg, December 2009.

[16] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, and N. Ferguson. On the Twofish key schedule. In *Proc. of the 1998 Selected Areas in Cryptography (SAC'98), Kingston, Ontario, Canada*, volume 1556 of *Lecture Notes in Computer Science*, pages 27–41. Springer, Berlin, Heidelberg, August 1998.

[17] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, and N. Ferguson. Twofish: A 128-bit block cipher. In *Proc. of the 1st Advanced Encryption Standard (AES) Conference, Ventura, California, USA*. NIST, June 1998.

[18] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, N. Ferguson, T. Kohno, and M. Stay. The Twofish team's final comments on AES selection. `https://www.schneier.com/academic/archives/2000/05/the_twofish_teams_fi.html` [Online; Accessed on May 1, 2017], May 2000.

[19] X. Zhao, S. Guo, F. Zhang, T. Wang, Z. Shi, and K. Ji. Algebraic differential fault attacks on LED using a single fault injection. Cryptology ePrint Archive Report 2012/347, 2012. `https://eprint.iacr.org/2012/347.pdf` [Online; Accessed on May 1, 2017].

[20] X. Zhao, S. Guo, F. Zhang, T. Wang, Z. Shi, C. Ma, and D. Gu. Algebraic fault analysis on GOST for key recovery and reverse engineering. In *Proc. of the 2014 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC '14), Busan, South Korea*, pages 29–39. IEEE, September 2014.

[21] X. Zhao, F. Zhang, S. Guo, T. Wang, Z. Shi, H. Liu, and K. Ji. MDASCA: An enhanced algebraic side-channel attack for error tolerance and new leakage model exploitation. In *Proc of the 3rd International Workshop on Constructive Side-Channel Analysis and Secure Design (COSADE'12), Darmstadt, Germany*, volume 7275 of *Lecture Notes in Computer Science*, pages 231–248. Springer, Berlin, Heidelberg, May 2012.

Author Biography

Chujiao Ma received the B.S. from Franklin W. Olin College of Engineering in 2010 and is currently pursuing her Ph.D. degrees in Computer Science and Engineering from University of Connecticut. She is a member of Comcast Center for Cybersecurity Innovation. Her research interests include network security, side-channel attacks and countermeasures.

John Chandy is a Professor and the Associate Head of the Electrical and Computer Engineering Department at the University of Connecticut. Prof. Chandy is also Co-Director of the Connecticut Cybersecurity Center, Interim Director of the UConn Center for Hardware Assurance, Security, and Engineering, and Co-Director of the Comcast Center for Cybersecurity Innovation. Prior to joining UConn, he had executive and engineering positions in software companies working particularly in the areas of clustered storage architectures, tools for the online delivery of psychotherapy and soft-skills training, distributed architectures, and unstructured data representation. His current research areas are in high-performance storage systems, reconfigurable computing, embedded systems security, distributed systems software and architectures, and multiple-valued logic. Dr. Chandy earned Ph.D. and M.S. degrees in Electrical Engineering from the University of Illinois in 1996 and 1993, respectively, and a S.B. in Electrical Engineering from the Massachusetts Institute of Technology in 1989.

Zhijie J. Shi is currently an Associate Professor of Computer Science and Engineering at the University of Connecticut. He received his Ph.D. degree from Princeton University in 2004 and his M.S. and B.S. degrees from Tsinghua University, China, in 1996 and 1992, respectively. He is a member of IEEE and ACM. His current research interests include hardware mechanisms for secure and reliable computing, side-channel attacks and countermeasures, embedded system designs, and sensor network security.