Acta Scientific COMPUTER SCIENCES

Volume 3 Issue 9 September 2021

Review Article

Methods for DPA and Branch Prediction Side Channel Attack Mitigation

Siddharth Shankar Swain*

Department of Computer Sciences, BITS Pilani, Pilani Campus, India

***Corresponding Author:** Siddharth Shankar Swain, Department of Computer Sciences, BITS Pilani, Pilani Campus, India.

Received: July 25, 2021

Published: August 05, 2021

Abstract

Differential power analysis side channel attacks uses the power consumed by different instructions when executed on a processor. Repeated experimentation by trying to encrypt the data with different keys. CPU instructions consume varying power, executes in varying lengths of time. Basic operation for any program executing on a processor involves switching of semiconductors. Semiconductors consume current while switching. Shape of this power consumption profile reveals activity. Comparison of these power profiles reveals processes happening and data consumed. There is also vulnerabilities in branch prediction [3] and speculative execution which is exploited by another class of side channel attacks. Our proposed solution directly acts at assembly instructions level, where we use the method of outlining to replace only the repeated sets of instructions with Turing complete instructions [1]. We replace the actual instructions with different Turing complete instructions to introduce randomness and also to eliminate the explicit jump or branch instructions. At the end we present a game theoretic approach to model such side channel attacks [2], with the aim to have a cost benefit analysis of these attacks and what should be the right approach for the user/defender in this zero sum-two player game.

Keywords: Differential Power Analysis; Branch Prediction Attacks; Turing Complete Instructions; Two Player Game

Introduction

Differential power analysis side channel attacks uses the power consumed by different instructions when executed on a processor. Repeated experimentation by trying to encrypt the data with different keys. Trying to find out how much power is consumed per instruction in encryption and decryption by oscilloscope power profiling. CPU instructions consume varying power, executes in varying lengths of time. Basic operation for any program executing on a processor involves switching of semiconductors. Semiconductors consume current while switching. Shape of this power consumption profile reveals activity. Comparison of these power profiles reveals processes happening and data consumed. Some countermeasures for it in literature involves introducing desynchronization in the encryption process so that the power traces do not align, adding intentional noise that is not Gaussian to randomly change representation of secret parameters [4], other techniques involve adding random delays, fake cycles and unstable clocking, also using filter circuitry [5]. There is also vulnerabilities in branch prediction [3] and speculative execution which is exploited by another class of side channel attacks. Some countermeasures for it includes using of CFI (Control flow integrity) technique to stop control flow hijacking attacks on the committed path [6], separating speculative data from committed data to isolate possible leakage, Intel and AMD suggests inserting serialization instructions like lfence to prevent loading secret data [7]. Most of the cybersecurity research focus on either presenting a specific vulnerability or proposing a specific defense algorithm to defend against a well-defined attack scheme. Al- though such cybersecurity research is important, few have paid attention to the dynamic interactions between attackers and defenders, where both sides are intelligent and will dynamically change their attack or defense strategies in order to gain the up- per hand over their opponents. The key idea is to model attackers/defenders to have multiple levels of attack/defense strategies that are different in terms of effectiveness, strategy costs, and attack gains/damages. Each player adjusts his strategy based on the strategies cost, potential attack gain/damage, and effectiveness in anticipating of the opponents strategy. We tackle both the above problems in the later section by providing some techniques for mitigation of such attacks and also forming a game theoretic model to cope with the constantly evolving security gam in the side channel domain. We also talk about the future directions on how to develop leakage resilient solutions (assuming there will always be a counter attack by the attacker for every security solution how can we leak as little unwanted information as possible but still not reveal the full actual information, i.e. security in presence of leakage of information) to these side channel attacks.

Proposed solution

Our proposed solution has the following parts for minimizing the possibility of such side channel attacks:

- Any function/procedure/implemented algorithm requested for processing on hardware can be broken down in its fundamental level into a sequence of assembly instructions.
- We use first a method called Outlining i.e. Replacing repeated sequences of instructions with calls to equivalent functions. Here we are first finding common/repeated group of instructions which occurs at multiple places of the assembly code. We substitute that by an equivalent function call and then we randomly change the function definition when calling at different places of the program.
- For instructions involving branches we substitute such common/repeated sequence of instructions with calls to equivalent function. And in these functions we replace the actual group of instructions with equivalent instructions that are Turing complete instructions like mov, subleq, subtract and branch if negative, subtract and branch if non zero. As a result in the function definition we can have only one of the Turing complete instruction like mov or subelq doing the same work as those repeated sequence of instructions but here we are having only a single instruction function.

Outlining method

Outlining method treats programs/instructions as strings and tries to find repeated sequence of instructions and replaces it with call to equivalent functions. This method has the following steps:

- Programs/instructions are strings.
- Finding repeated substrings.
- Suffix tree data structure can be used to find these repeated substring.
- Ukonen's suffix tree construction algorithm can be used here.
- Finding repeated substrings is equivalent to find repeated instructions in the program.
- Once repeated instructions are found, we can replace them with equivalent function calls. So next step is inserting function calls at places where these repeated instructions are found.
- Last step is the function definition of the above called functions.

The method can be illustrated with the help of below example:

Let the program P1 has two functions namely Foo() and Bar() can be:

Func1()

A → R1 = 0xDEADBEEF

B →R3 = R2 + R1

C →R1 = ∗R5

D →R7 = 0xDEEFFACE

E →R1 = R1 -1

Func2()

F →R7 = R3 + R1

A → R2 = 0XDEADBEEF

B →R3 = R2 + R1

C →R1 = ∗R5

G → R7 = OxFEEDDEAF

A → R2 = 0XDEADBEEF

B →R3 = R2 + R1

C →R1 = ∗R5

E →R1 = R1 -1

Here the longest repeated sub string is ABC with length of 3.

As can be seen in the example we encode each instruction in the left as an alphabet. And we get a string out of it like - A B C D E $0 F A B C G A B C E $1.

The $ are inserted to separate group of strings of various instructions. Our task is to find the longest repeating sub string in the entire string. This longest repeated sub string corresponds to the longest common repeated set of instructions. This is accomplished in the most efficient manner by using suffix tree data structure to represent the above string and finding longest repeated common substring from it. Ukonen's suffix tree construction algorithm is used for this purpose. Here we are inserting the function definition which corresponds to the longest repeated set of instruction and give our function a name (here OUTLINED()). In th previous example we found the longest common sub string i.e. ABC. So the Outlined function can be like (taking th previous example):

OUTLINED()

A → R2 = 0XDEADBEEF

B → R3 = R2 + R1

C → R1 = ∗R5

Here we insert function calls instead of the repeated set of instructions. In our case we in the function definition (here OUTLINED()) we replace the original instructions with Turing complete instructions like mov, subelq etc.as any instruction can be replaced by its equivalent Turing complete instructions.

Note

The above one is just an example of Outlining method and has already mov instruction. Actual instruction will include more variety and types of instructions which will be eventually replaced by equivalent Turing complete instructions.

Turing complete instructions

Turing complete instructions are those that can be called the ultimate reduced instruction where they are used to represent any number and type of instructions available in the ISA. In short they are complete in the sense that, a single instruction can be used to represent any number and type of instructions in the architecture. The most common example is "mov" instruction in x86 architecture. Any instruction in x86 arch can be substituted by equivalent "mov" instructions without com- promising the function and correctness of the original instruction. Any code we write, can be written as a set of "mov" s instead. So lets look at some Turing complete instructions and how we can convert instructions to its Turing complete counterparts:

Let take the equality operation. Suppose we have two numbers say 'x' and 'y' and we want to figure out if X == Y. So for this we have ready made instructions in our ISA, but how to do this check with only "mov" instructions.

Let us suppose X and Y are memory addresses here. So the above operation can be replicated by equivalent mov operation as follows:

Case-1: Let X = 2 and Y = 3

mov [X], 0

mov [Y], 1

mov R, [X]

Here lets see for each instruction:

mov [X], 0 → move 0 to memory location X, i.e. move 0 to memory location 2 mov [Y], 1 → move 1 to memory location Y, i.e. move 0 to memory location 3. mov R, [X] → move 0 to memory location R.

Now by seeing the value at memory location R we can know if X and Y are equal, i.e. if R has 0 then they are not equal if R has 1 they are equal. Here value at R is 0 so X!=Y.

Case-2: Let X = 3 and Y = 3

mov [X], 0

mov [Y], 1

mov R, [X]

Here lets see for each instruction: mov [X],

0 → move 0 to memory location X, i.e. move 0 to memory location 3 mov [Y], 1 → move 1 to memory location Y, i.e. move 1 to memory location 3. mov R, [X] → move 1 to memory location R.

Now by seeing the value at memory location R we can know if X and Y are equal, i.e. if R has 0 then they are not equal if R has 1 they are equal. Here value at R is 1, so X == Y.

Let's see another Turing complete instruction like "subleq" - subtract and branch if less than or equal to zero. This can be explained in form of memory operations as follows:

subleq a, b, c

Mem [b] = Mem [b] - Mem [a];(equivalent memory operation)

if (Mem [b] ≤ 0) goto c

Similar to "mov" instruction we can substitute "subleq" instead of any other instruction. It does the same work. For example.

For unconditional jump instructions like "JMP A", can be substituted by:

subleq R, R, A

"ADD b, c" can be substituted by:

subleq b, Rsubleq R, csubleq R, R

So there are a number of Turing Completeinstructions like this, we can list them as follows:

- Subtract and branch if less than or equal to zero
- Subtract and branch if negative
- Reverse subtract and skip if borrow
- Move
- Subtract and branch if non zero (SBNZ a, b, c,destination).

Actual algorithm flow

Below is the flow diagram for the algorithm:

Let's go through an example to show the actual algorithm flow:

callq printf

movl 2, edx

movl -8(rbp), esi

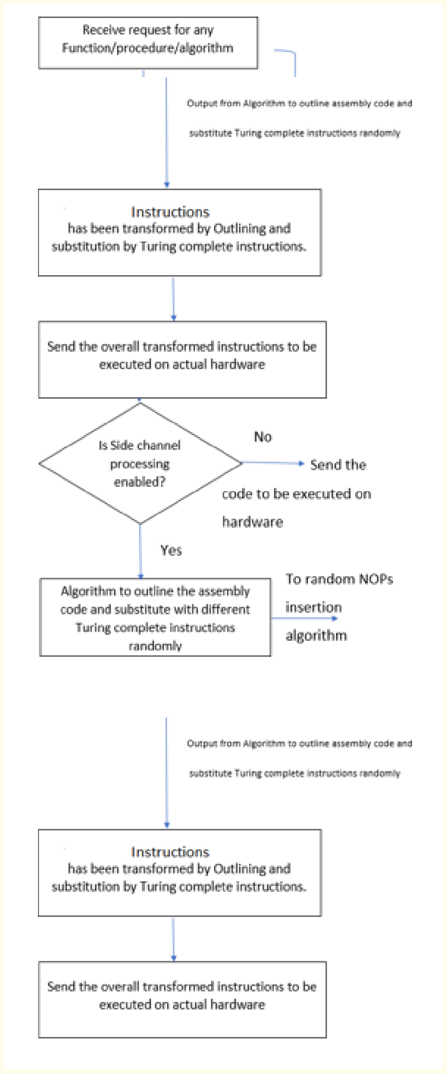

Receive request for any Function/procedure/algorithm
Output from Algorithm to outline assembly code and substitute Turing complete instructions randomly
Instructions has been transformed by Outlining and substitution by Turing complete instructions.
Send the overall transformed instructions to be executed on actual hardware
Is Side channel processing enabled?
No
Send the code to be executed on hardware
Yes
Algorithm to outline the assembly code and substitute with different Turing complete instructions randomly
To random NOPs insertion algorithm
Output from Algorithm to outline assembly code and substitute Turing complete instructions randomly
Instructions has been transformed by Outlining and substitution by Turing complete instructions.
Send the overall transformed instructions to be executed on actual hardware

Figure a

addl 1, esi

movl esi, -8(rbp)

...

movl -20(rbp), ecx

movl 2, edx

movl -8(rbp), esi

addl 1, esi

```
movl esi, -8(rbp)
...
movl 2, edx
movl -8(rbp), esi
addl 1, esi
movl esi, -8(rbp)
```

This is the code we get for a simple function/procedure after the pre processing stagein the flow diagram.

Next, we check whether side channel processing is enabled for this code (this is to be decided by the software developer according tohis requirements of speed and security).

If (enabled) then we go to the next step otherwise just send the generated/pre processed code to be executed on actual hardware.

Next step - We explain the algorithm flow using the above example. the above assembly code for the function can be represented by individual characters like:

```
A :callq printf
B :movl 2, edx
C :movl -8(rbp), esi
D :addl 1, esi
E :movl esi, -8(rbp)
...
F :movl -20(rbp), ecx
B :movl 2, edx
C :movl -8(rbp), esi
D :addl 1, esi
E :movl esi, -8(rbp)
...
B :movl 2, edx
C :movl -8(rbp), esi
D :addl 1, esi
E :movl esi, -8(rbp)
```

From the above encoding we can usethe Ukkonen's suffix tree construction [8] algorithm to find the longest repeated sub- string (which is "BCDE" in this case). So after finding longest repeated substring the longest repeated set of assembly instruction can be found as like.

Like this in the example:

```
callq printf
callq func()
...
movl -20(rbp), ecx
callq func()
...
movl 2, edx
callq func()
callq printf
movl 2, edx
movl -8(rbp), esi
addl 1, esi
movl esi, -8(rbp)
...
movl -20(rbp), ecx
movl 2, edx
movl -8(rbp), esi
addl 1, esi
movl esi, -8(rbp)
...
movl 2, edx
movl -8(rbp), esi
addl 1, esi
movl esi, -8(rbp)
```

Then we define a function to execute these repeated instructions and replace these instructions with the appropriate function call.

```
func():
movl 2, edx
movl -8(rbp), esi
```

```
addl 1, esi
movl esi, -8(rbp)
retq
```

In the next step we replace the repeatedinstructions with appropriate function call.

In the next step we insert equivalent Turing complete instructions as described in the above sections in the function definition instead of real instructions. We listed above some Turing complete instructions, butwe will show this example with "subleq" instruction.

```
func():
movl 2, edx
movl -8(rbp), esi
addl 1, esi
movl esi, -8(rbp)
retq
```

The above function can be represented as:

The content at location Z (is 0) initially

```
func();
subleq edx, edx
subleq 2, Z
subleq Z, edx
subleq Z, Z
subleq esi, esi
subleq -8(rbp), Z
subleq Z, esi
subleq Z, Z
subleq 1, Z
subleq Z, esi
subleq Z, Z
subleq -8(rbp), -8(rbp)
subleq esi, Z
subleq Z, esi
subleq Z, Z
retq
```

This will confuse the attacker as in his DPA analysis of these instructions he will not get the actual mapping of the power with the actual instruction executed in the functionas it is being hidden with a Turing complete instruction.

Similar to "subleq" the instructions can be substituted by other Turing complete instructions in the list in the same way. And each time we substitute these instructionswe choose a different Turing complete instruction to substitute the actual instructionwhich further adds to the randomness makesthe DPA analysis more harder.

In the next step we send the overall trans- formed instructions to be executed on actualhardware.

The above example is given for unconditional instructions. Similar procedure can be followed for handling conditional instructions. the point tonote here is that the actual call in unconditionalor conditional instructions can be replaced byTuring complete instructions itself which re- duces the security flaws that can be exploited by branch prediction techniques (as there will beno branches here, so no branch prediction). But having said that, replacing every instruction in theentire program with Turing complete instructionswill increase the overall code size, hence time and memory instructions considerably, so we have developed a strategy here to only use Turingcomplete instructions with outlined code. This will somewhat break the pattern as outlined code are the repeated ones and create patterns in DPA attacks to be exploited and also using it only withoutlined code help us to control the code size increase. So it increases security of our systems against side channel attacks without considerablyincreasing the code size and time of execution.

Game theoretic approach

Most of the cybersecurity research focus on either presenting a specific vulnerability or proposing a specific defense algorithm to defend against a well-defined attack scheme. Although such cybersecurity research is important, few have paid attention to the dynamic interactions between attackers and defenders, where both sides are intelligent and will dynamically change their attack or defense strategies in order to gain the upper hand over their opponents. The key idea is to model attackers/defenders to have multiple levels of attack/defense strategies that are different in terms of effectiveness, strategy costs, and attack gains/damages. Each player adjusts his strategy based on the strategy's cost, potential attack gain/damage, and effectiveness in anticipating of the opponent's strategy. We study the achievable Nash equilibrium [9] for the attacker-defender security game where the players employ an efficient strategy according to the obtained equilibrium. We will analyze the side channel attacks in terms of two player zero sum game, where this dynamic modelling will help us to accurately consider the behaviour of the players involved in side channel attacks and protection game.

This can be formulated as a non-cooperative zero-sum game. In addition, we describe at- tacker and defender strategies and derive their solutions. Being rational players in the game, an attacker competes for the best action and his objective is to maximize his own utility. Therefore, the opponents are not bound to cooperate with each other where the malicious attacker would want to play a suitable strategy to maximize his chances of being successful and waste the resources of the system. In contrast, the defender would also like to play a suitable strategy to maximize his chances of protection againstthe opponents without overspending energy or computation on defending.

Game Model - We consider two player non-coordination zero sum game for the side channel attack scenario. It can be represented by:

- G = < (P), (S), (U) >, represents the two player game model in side channel attacks.
- P = A, {D}, represents two players in the game i.e. the attacker - A and the defender - D.
- S = {0, 1, 2}, represents the strategy space, set of actions available for each player. O rep- resents no attack or no defense from both the players, 1 represents a low intensity attack or defend, 2 represents a high intensity attack or defense. We can increase the number of strategies here depending upon the complexity of the side channel attack.
- U here refers to the expected pay- off/profit/motivation for playing the game by each player.

Attacker and defender have three options/actions(A) each:

- a_0 = No attack, a_1 = Low intensity attack, a_2 = High intensity attack
- d_0 = No defense, d_1 = Low intensity defense, d_2 = High intensity defense.

In any type of side channel attack, there is some value of protected assets by defender. Let the value of the asset be $v_n > 0$, where $n\epsilon$ 1, 2 Here v_1 value of assets compromised by Attack - 1 strategy deployed by attacker successfully.

v_2 →value of assets compromised by Attack - 2 strategy deployed by attacker successfully.

v_n →is the gain of the attacker if his strategy Attack -n is successfully. Also it is the asset loss of the defender if the attack is successful.

The attacker/defender also need to make some effort to implement their attack/defense strategies,For attacker represented by C_{an}, $n\epsilon$ 1, {2}

C_{a1} →Cost to deploy Attack - 1.

C_{a2} →Cost to deploy Attack - 2.

Similarly for the defender represented by

C_{dn}, $n\epsilon\{1, 2\}$

C_{a1} →Cost to deploy Defense - 1.

C_{a2} →Cost to deploy Defense - 2.

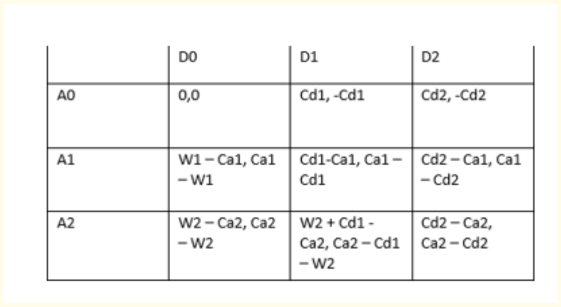
In addition, the game model requires us to define what is the outcome when the attackerdeploys one specific attack strategy and the de- fender implements one specific defense strategy. We make the following assumptions on the gameoutcomes.

Attack is successful under these scenarios: Attack-1 vs. No-Defend; Attack-2 vs. Defend-1 or No-Defend.

Defense is successful under these scenarios: Defend-1 vs. Attack-1 or No-Attack ; Defend-2vs. Attack-2 or Attack-1 or No-Attack.

Zero gain or loss when there is no attack and nodefense deployed, i.e., No-Attack vs. No-Defend.

The above assumptions mean that the more aggressive defense strategy, Defend-2, is secure against all attacks. However, the low-level defense strategy, Defend-1, is good to defend the low-level attack, Attack-1, but is still vulnerable to deal with the aggressive attack, Attack-2. Figure b illustrates the payoff matrix of the game in a strategic form.



	D0	D1	D2
A0	0,0	Cd1, -Cd1	Cd2, -Cd2
A1	W1 – Ca1, Ca1 – W1	Cd1-Ca1, Ca1 – Cd1	Cd2 – Ca1, Ca1 – Cd2
A2	W2 – Ca2, Ca2 – W2	W2 + Cd1 - Ca2, Ca2 – Cd1 – W2	Cd2 – Ca2, Ca2 – Cd2

Figure b

Here the above figure describes the utility or payoff the attacker or defender payoff or gain de- pending on the attacker or defenders move. This is the payoff when players have pure strategies i.e. (A pure strategy determines all your moves during the game (and should therefore specify your moves for all possible other players' moves). For mixed strategy (A mixed strategy is a probability distribution over all possible pure strategies (some of which may get zero weight). After a player has determined a mixed strategy at the beginning of the game, using a randomising device, that player may pick one of those pure strategies and then stick to it). For mixed strategy payoffs of the attacker and defender we can find the expected utility of each of the players considering he will make one move or the other.

Mixed strategy is a probability distribution over pure strategies. Let the probabilities associated with pure strategies be P = (p1, p2, p3,..., pr) and mixed strategy is a probability distribution over pure strategies:

$$\sum_{t=1}^{R} p_t = 1$$

p_{a0} →Probability (playing strategy a_0)

p_{a1} →Probability (playing strategy a_1)

p_{a2} →1 -p_{a0} -p_{a1} Probability (playing strategy a_2)

In the same manner for defender:

p_{d0}, p_{d1}, p_{d2}

Expected Utility for attacker:

$$EU(p_{a0}) = p_{d0}(0) + p_{d1}(-C_{d1}) + p_{d2}(-C_{d2})$$

$$EU(p_{a1}) = p_{d0}(w_1 - C_{a1}) + p_{d1}(C_{d1} - C_{a1}) + pd2(Cd2 - Ca1)$$

$$EU(p_{a2}) = p_{d0}(w_2 - C_{a2}) + p_{d1}(w_2 + C_{d2} - C_{a2}) + p_{d2}(C_{d2} - Ca2)$$

Expected Utility for defender:

$$EU(p_{a0}) = p_{a0}(0) + p_{a1}(C_{a1} - w_1) + p_{a2}(C_{a2} w_2)$$

$$EU(p_{a1}) = p_{a0}(C_{d1}) + p_{a1}(C_{a1} - C_{d2}) + p_{a2}(C_{a2} - w_2 - C_{d1})$$

$$EU(p_{d2}) = p_{a0}(C_{d2}) + p_{a1}(C_{a1} - C_{d2}) + pa2(Ca2 - Cd2)$$

This model can be used for different side channel attack scenarios and the expected utility calculated can be used to help the user to take a safe place in the continuous defender attacker game which is played in the security domain. There is a continuous evolution of both the sides to win this game and there can be various mitigation techniques for mitigating these attacks but not completely eliminating it. So game theory approach applied to side channel attack scenarios can be interesting to explore. Here in the game theoretic approach we have provided a very generalized framework which can change (like the attack and defence levels, the strategies used in the attack) according to the variety of parameters and constraints imposed by the side channel attacks.

Conclusion

The outcome and result of this technique is a compiler which compiles according to the above mentioned algorithm and generate Turing complete instructions without increasing the code size considerably by using the outlining method mentioned in the paper, other outcome of this paper is using game theoretic model of analysis to suggest future solutions for side channel attacks. Such model can help to build effective and correct countermeasures for the side channel attacks. Differential power analysis and branch predictionside channel attacks are hard to eliminate, it can only be mitigated to some extent. They are there in our hardware because of fundamental techniques used to build the hardware and also speed up the processing in these hardwares. So eliminating these age old techniques completely is not a goodsolution, but how can we exist with these techniques still protect ourselves from such type of attacks should be the area of research. Our techniques discussed in this paper increases security of our systems against side channel attacks with- out considerably increasing the code size and timeof execution. So this is just a mitigation techniquewhich decreases the vulnerabilities of such systemand makes it less prone to attacks. Just taking forgranted whatever solution you provide will havea counterattack by the attackers in near future wehave to make the system secure by the game theoretic approach of analysis which will help to determine what all strategies we as players can take with the constant evolution of the attackers. Future research can lead to how to use game theory in such scenarios more effectively also how to develop leakage resilent solutions (assuming there will always be a counter attack by the attacker forevery security solution how can we leak as little unwanted information as possible but still not re-veal the full actual information, i.e. security in presence of leakage of information) to these sidechannel attacks.

Bibliography

1. https://www.cl.cam.ac.uk/sd601/papers/mov.pdfAPA:83
2. Standaert Franois-Xavier. "Introduction to side-channel attacks". Secure Integrated Circuits and Systems. Springer, Boston, MA (2010): 27-42.
3. Aciimez Onur., *et al*. "On the power of simple branch prediction analysis". Proceedings of the 2nd ACM symposium on Information, computer and communications security. ACM (2007).
4. Kocher Paul C. "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems". Annual International Cryptology Conference. Springer, Berlin, Heidelberg (1996).
5. Shamir Adi. "Protecting smart cards from passive power analysis with detached power supplies". International Workshop on Cryptographic Hardware and Embedded Systems. Springer, Berlin, Heidelberg, (2000).
6. Koruyeh Esmaeil Mohammadian., *et al*. "Speccfi: Mitigating spectre attacks using cfi informed speculation". 2020 IEEE Symposium on Security and Privacy (SP). IEEE (2020).
7. ADVANCED MICRO DEVICES, INC. Software techniques for managing speculation on and processors (2018).
8. Ukkonen Esko. "On-line construction of suffix trees". *Algorithmica* 14.3 (1995): 249-260.
9. Nash John. "Non-cooperative games". *Annals of Mathematics* 54.2 (1951): 286-295.

Volume 3 Issue 9 September 2021