IMPROVING THE ENERGY-BASED DPA/DEMA ATTACK FLOW PREPROCESSING PERFORMANCE

Rodrigo Nuevo Lellis[1], **Rafael Iankowski Soares**[1]

[1]Group of Architectures and Integrated Circuits (GACI) – UFPel – Pelotas, RS, Brazil

{rn.lellis,rafael.soares}@inf.ufpel.edu.br

Abstract. *This paper proposes an improvement in the DPA/DEMA energy-based attack flow to reduce the time to guess a secret cryptographic key from supposed secure systems. This was done through the recoding for C ++ language and parallelization of the algorithms. The results highlight a reduction of up to 78.53% in the execution time of preprocessing algorithms guaranteeing a good performance even in the majority of off-the-shelf processors available in the market.*

1. Introduction

DPA/DEMA attacks are very sensitive to the alignment of power traces. This led to the development of several countermeasures based on the misalignment of these traces [CORON and KIZHVATOV 2009]. However, preprocessing steps like the one proposed by [LELLIS et al. 2017] can neutralize such measures. The attack flow of [LELLIS et al. 2017] aims to realign the traces through 3 steps: (i) extracting the target signature, (ii) subsampling traces, and (iii) energy calculation. These algorithms were initially implemented in Matlab [MATHWORKS] and compute a large amount of traces, i.e., power consumption acquired during the execution of the system, to obtain a successful attack.

The implementation available in Matlab is costly at runtime due to the use of the virtual machine. For this reason, the translation of the algorithms into a compiled language such as C++ is essential to reduce the execution time of the preprocessing steps. In addition, the preprocessing algorithms of [LELLIS et al. 2017] have many loops that can be parallelized by programming directives that divide the workload into different threads to be performed [DURAN et al. 2005]. Thus, the directive parallel for, available in the GCC compiler [GNU COMPILER COLLECTION], through the OpenMP library [OPENMP], was used to explore the parallelism present in the algorithms.

2. Experiments and Results

The experiments are performed on a computer with a CPU Intel Core i5 3317U processor, 1.7GHz, and 8GB RAM. The processor has two physical cores, which are associated with Intel Hyperthreading, can run up to four independent threads.

Table 1 summarizes the runtime results of the algorithms developed in the C++ language with and without the use of high-performance processing techniques.

All experiments in Table 1 are performed 5 times. Thus, we have in the first row the average execution time in seconds for each of the attack flow steps. The second line contains the standard deviation of the executions. It is possible to see that the standard

Table 1. Runtime Results

	Extract Signature				Subsampling				Energy			
	Matlab	**C++**	**2 Thds**	**4 Thds**	**Matlab**	**C++**	**2 Thds**	**4 Thds**	**Matlab**	**C++**	**2 Thds**	**4 Thds**
Avg.	14284	6663	4084	3067	6853	4329	2283	2109	2428	6968	2285	1687
S.D.	-	181	364	122	-	248	31	184	-	184	63	65
R.T.	78.53%				69.23%				30.52%			

deviation for all experiments is small, which indicates that the average is reliable. The last line of Table 1 represents the the maximum percentage reduction in terms of execution time, in relation to the Matlab implementation [MATHWORKS] of the algorithms. We can see that, the extract signature step has a time reduction of 78.53% when executing on 4 threads. This significant reduction is justified by the parallelism present in the algorithm. Likewise, the other algorithm steps have also obtained significant reductions in terms of execution time.

3. Conclusions

This paper proposes to recode the algorithms of [LELLIS et al. 2017] in a compiled language C ++, with the use parallelization techniques. The purpose of this implementation is to reduce the processing time of the attack flow steps, which are computationally costly due to the huge amount of traces required to perform the attacks. From the results, can be seen that all stages of the attack flow have their execution times reduced, reaching a maximum reduction of 78.53%. This leads us to conclude that optimizations in the preprocessing steps for DPA/DEMA attacks make attackers much more efficient, making them able to carry out attacks much faster if they use high-performance processing techniques. Thus, it becomes increasingly important to develop new countermeasures preventing attacks even with preprocessing steps included in the flow.

References

CORON, J. and KIZHVATOV, I. (2009). An Efficient Method for Random Delay Generation in Embedded Software. Clavier C., Gaj K. (eds) Cryptographic Hardware and Embedded Systems (CHES).

DURAN, A., GONZÀLEZ, M., and CORBALÁN, J. (2005). Automatic Thread Distribution For Nested Parallelism In OpenMP. pages 121–130. 19th ACM International Conference on Supercomputing (ICS).

GNU COMPILER COLLECTION. GCC online documentation – GNU Project – Free Software Foundation. https://gcc.gnu.org/onlinedocs/. Online; accessed: 7 December 2019.

LELLIS, R. N. ., SOARES, R. I., and JUNIOR, A. A. S. (2017). An Energy-Based Attack Flow for Temporal Misalignment Countermeasures on Cryptosystems. IEEE International Symposium Circuits and Systems (ISCAS).

MATHWORKS. MATLAB Support Documentation. https://www.mathworks.com/help/signal/ug/resampling.html. Online; accessed: 6 December 2019.

OPENMP. Specifications – OpenMP. https://www.openmp.org/specifications/. Online; accessed: 8 December 2019.