Modelling the Impact of Code Obfuscation on Energy Usage

Athul Raj, Jithish J and Sriram Sankaran
Amrita Center for Cybersecurity Systems and Networks
Amrita School of Engineering, Amritapuri
Amrita Vishwa Vidyapeetham
Amrita University
India
srirams@am.amrita.edu

ABSTRACT

Advancements in computing and communication technologies have given rise to low-cost embedded devices with applications in diverse domains such as Smarthome, industrial automation, healthcare, transportation etc. These devices are power-constrained which emphasizes the need for lightweight security solutions. Code obfuscation has been demonstrated to provide time-limited protection of source code from inference or tampering attacks. However, size of the obfuscated code increases with increase in code size which can have a negative impact on energy consumption. In particular, different transformations of the source code result in varying amounts of energy consumption for embedded devices. In this work, we model the impact of algorithms for code obfuscation on energy usage for embedded devices and analyze the energy-security-performance trade-offs. The insights from our analysis can be used to develop techniques depending on the needs of the applications thus facilitating efficient energy usage.

1. INTRODUCTION

The proliferation of low-cost embedded devices with advancements in computing and communication capabilities facilitates device-to-device communication with varying capabilities. Energy Management has become one of the foremost concerns in mobile devices. This is primarily due to the increasing functionality in applications which rapidly drains battery power in these devices. In addition, security contributes towards energy drain due to the significant performance overhead of mechanisms used to secure embedded devices. These emphasize the need for developing energy aware security mechanisms for embedded devices. Further these mechanisms require measurement tools for estimating the energy consumed due to security for embedded devices.

Due to the increasing sensitivity of software applications stored on embedded devices, mechanisms are necessary for protecting their source code which runs on these devices. Code obfuscation has been shown to provide time-limited protection of source codes thus preventing intruders from tampering or inference attacks. The primary role of a code obfuscator is to apply a transformation to the original source code thus resulting in obfuscated source code which in turn is stored in the embedded devices. This results in obstructing the regular control flow and performing software manipulations thus making it impossible to reverse engineer the source code. In addition, code obfuscation has a considerable impact on performance and energy consumption which varies for different kinds of applications.

While code obfuscation has been demonstrated to provide time-limited protection for embedded devices, its impact on energy consumption needs to be analyzed. In particular, increase in size of the source code results in corresponding increase in size of the obfuscated code thus causing a negative impact on power consumption. Further different transformations of the source code result in varying amounts of energy consumption for embedded devices. Thus a comprehensive energy analysis of algorithms for code obfuscation is necessary to analyze the energy-performance trade-offs. In addition, estimating security is necessary to analyze the effectiveness of the obfuscation techniques. However, estimating security of obfuscated code is challenging due to the varying number of transformations applied to source codes with increasing functionality.

In this work, we model the impact of algorithms for code obfuscation on energy usage in embedded devices. In particular, we study different techniques for obfuscation such as Lexical, Data and Control obfuscation and analyze their impact on energy usage. We describe the experiment set-up for measuring energy and performance and present our analysis of energy-performance trade-offs. Our analysis conducted using Mibench benchmarks reveals that data and control flow obfuscation incur significant energy consumption compared to lexical obfuscation. The results obtained from this study can be used to tailor obfuscation to suit the needs of resource-constrained devices and further analyze energy-security-performance trade-offs.

2. RELATED WORK

Numerous approaches have studied the problem of code obfuscation and analyzed their applicability in embedded devices. Collberg *et. al* [9] presented a taxonomy of different kinds of obfuscation which describes transformations for

securing diverse source codes. Further, there exists tools for code obfuscation at both the hardware and software levels. While some of the tools such as Tigress [8] and ProGuard [18] are open-source, there exists numerous commercial tools such as Allatori [1], Dasho [2], Zelix [5].

There exists approaches for measuring the impact of obfuscated code on energy consumption. In particular Sahin *et al.* [21] profiled the energy consumption of different transformations on the Android smartphone and statistically analyzed the significance between normal and obfuscated code. However, they claimed that energy consumption between different methods of a particular kind of obfuscation is insignificant. In addition, Dukovic *et al.* [11] profiled the energy consumption of normal and obfuscated code at an instruction-level and analyzed the impact of different transformations.

In addition to profiling energy consumption, certain approaches have analyzed the security of obfuscated code for diverse kinds of systems. Banescu *et al.* [6] measured the resilience of obfuscated code against reverse engineering using tools such as tigress which contains numerous transformations. Wu *et al.* [26] estimated the security of different obfuscation based techniques using approximation. In particular, their approach involves modelling the relationship between obfuscation parameters and the corresponding impact on security using regression-based techniques.

While energy and security are critical, performance is equally necessary so as to meet real-time constraints for embedded applications. This can be achieved using hardware obfuscation which is necessary to protect secret information in circuit design. Kainth *et al.* [16] have developed a hardware-assisted technique for code obfuscation for FPGA based microprocessors. Their approach involves modelling the transformations on the FPGA so as to improve their performance as well as enhance the security of the applications. Further, the reprogrammable nature of the FPGAs makes the overall design of techniques adaptable.

In addition, numerous works have modelled the energy consumption of computing systems in general and embedded devices. Economou *et al.* [12] and Isci *et al.* [15] modeled the energy consumption of embedded devices using performance counters. Khan *et. al* [17] and Sankaran *et al.* [23] modeled the energy consumption of multi-core systems using a statistical learning approach. Wang *et al.* [25] developed SPAN, a software power monitoring tool which correlates program segments with power consumption. Fan *et al.* [13] analyzed the power consumption characteristics of data centers and studied the optimal provisioning of resources. Sangaiah *et al.* [22] developed regression models to predict the performance of Chip Multiprocessors.

In contrast to the existing approaches, we profile the energy consumption and performance of algorithms for code obfuscation in embedded devices. Although our work is closely related to [11] and [21], we manually obfuscate source code of benchmark applications which enables us to explore numerous transformations in contrast to the obfuscation tools used by [11] and [21]. While tools for obfuscation are available, they are often commerical and that we are limited by the transformations available in those tools. In addition, our experiments are conducted using SourceMeter, a power measurement tool which provides accurate power estimates.

3. BACKGROUND

3.1 Lexical Obfuscation

Lexical obfuscation is one of the basic and simplest form of obfuscation used in software programs. It includes a wide array of operations such as comment removal, identifier renaming, structured construction removal, debugging info removal etc. Typically lexical obfuscation is used for identifier renaming which may not have any impact on the security of the source code. Thus it becomes relatively easy for the attacker to understand and reverse-engineer the source code. In addition, other techniques such as structured construction removal, debugging info removal and comment removal may reduce the size of the source code. While lexical obfuscation lacks security, it has been either replaced or used in conjunction with other techniques such as data and control obfuscation.

3.2 Data Obfuscation

Data obfuscation provides stronger security than lexical obfuscation by protecting data in the source code. In particular, data is protected in such a way such that it becomes hard to infer the functionality through code analysis. Data obfuscation includes a wide array of operations such as string scrambling, array restructuring and merging, data encoding and variable reordering. For instance, value of a variable can be changed to include numerous different variables. Thus the value of the variable can be determined by fusing the contents of the created variables. Similarly, arrays can be restructured by creating arbitrary number of new arrays and merging them with existing ones. Techniques for data obfuscation have been described in [10] and [20]. Thus by combining different kinds of techniques for data obfuscation, overall security can be enhanced. However, size of the obfuscated code may increase which causes a negative impact on power and execution time.

3.3 Control Obfuscation

Control obfuscation obfuscates the control flow of the program thus providing stronger security than lexical and data obfuscation. This includes manipulating the flow of execution with irrelevant conditional statements which in turn results in restructuring of methods, loops and statements. Typically, restructuring consists in inlining, outlining, interleaving and cloning of functions and elimination of library calls [24] [7]. In addition, a single function is transformed through opaque predicates, insertion of dead code etc. However, as the control flow of the program is obfuscated, it becomes harder for the attacker to interrelate the various sections of the program. As a result, there is an increase in the size of the obfuscated code which negatively impacts power consumption and execution time.

4. EXPERIMENT SET-UP

In this section, we discuss the hardware and software used for our experiments along with the description of our power measurement set-up for energy analysis. We use the STM32-F0DISCOVERY [19] development board from STMicroelectronics as the embedded platform for measuring the energy impact of code obfuscation. The board operates at a DC power supply of 3V. The board consists of an ARM Cortex-M0 microcontroller with 64KB flash and 8KB RAM. In addition, Keil MDK-ARM embedded software development envi-

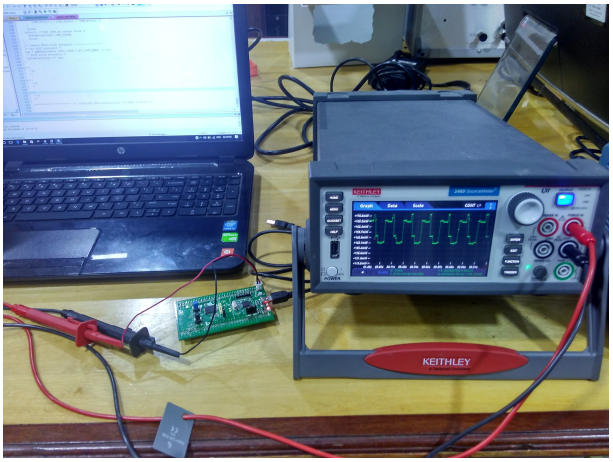

Figure 1: Experiment Set-up

ronment [3] is used to develop, compile and debug the soure code as well as flash the program on the development board for evaluating different obfuscation techniques.

***Power Measurement*:**

Keithley's Series 2400 Source Measure Unit (SMU) [4] is used for measuring the power consumption of the board. It integrates both source and measurement circuitry in a single unit thus facilitating a fast and accurate measurement of power consumption. Figure 1 contains the pictorial representation of the set-up used for our experiments.

The SMU is used as the DC power source for the set-up.It provides the required stable precision DC power supply of 3V for the embedded board. The obfuscated programs are developed on the Keil MDK-ARM software suite running on Windows 10 platform and flashed to the board via USB interface. The SMU logs the power consumed by the embedded board at distinct time intervals. Further we perform comparative analysis of power traces using MATLAB.

***Benchmark Applications*:**

We consider the following applications from the Mibench benchmark suite [14] for comparatively analyzing the impact of code obfuscation on energy usage. Similarly other applications can be profiled and its energy impact on code obfuscation analyzed.

Basicmath: Mathematical calculations invoving cubic function solving, square root evaluation, degrees to radian conversion etc. on a fixed set of constants.

Bitcount: Evaluating bit manipulation capabilities by computing the total number of bits in a fixed input array of integers.

Matrix Multiplication: Computing the matrix multiplication which is of complexity $O(n^3)$.

Vernam Cipher: Symmetric stream cipher which utilizes Boolean XOR operation to generate the ciphertext.

5. ENERGY ANALYSIS

In this section, we analyze the energy consumption of different code obfuscation techniques using the embedded benchmarks. In particular, we consider obfuscation techniques such as lexical, data and control flow transformations. Within the context of data obfuscation, we transform array data [20] for code obfuscation. In control flow obfuscation, we perform control flow flattening, dead code insertion, extended loop condition and loop transformation. Further, we analyze a transformation which involves a combination of above obfuscation techniques termed as combined obfuscation.

Table 1: Lexical, Data and Combined Obfuscation

App	**%ΔP**	**%ΔT**	**%ΔE**	**%ΔS**
Matrix	LO:1.15 DO:1.5 CO:1.51	LO:0 DO:1.5 CO:8	LO:1.14 DO:0 CO:9.63	LO:0 DO:4.51 CO:21.47
Basicmath	LO:0.698 DO:2.29 CO:4.13	LO:2.94 DO:8.82 CO:8.82	LO:3.66 DO:11.32 CO:13.32	LO:0 DO:0.99 CO:10.04
Bitcount	LO:0.93 DO:0.18 CO:0.124	LO:0 DO:2.81 CO:14.1	LO:2.36 DO:3.01 CO:13.8	LO:0 DO:0.51 CO:16.4
Vernam	LO:0 DO:0.25 CO:1.32	LO:0 DO:4.16 CO:12.6	LO:0 DO:4.42 CO:13.99	LO:0 DO:2.51 CO:18.1

LO:Lexical Obfuscation DO:Data Obfuscation
CO:Combined Obfuscation

In our experiments, we manually obfuscated the source code of the applications using the above transformations. While there exists open-source and commercial tools for code obfuscation, they are limited in terms of the number of available transformations. Thus, our goal is to explore numerous transformations and their possible combinations and not be restricted by the tools.

Our analysis reveals that different operations performed by the Cortex M0 microcontroller have corresponding power consumption profiles. Thus, our goal is to analyze the impact of different obfuscation techniques on the energy and performance of embedded devices. Towards this goal, we gather the power traces from the SMU for normal and obfuscated codes pertaining to different transformations. Further, execution time and energy consumption were estimated for each of the applications. In particular, we compute the percentage difference of parameters such as average power, execution time, energy consumption and storage between normal and obfuscated code.

Table 1 contains the power, execution time, energy and storage results for lexical, data and combined obfuscation while table 2 displays those for each of the transformations in control flow obfuscation. From the table, %ΔP, %ΔT, %ΔE and %ΔS denote the percentage difference of parameters such as power consumption, execution time, energy and storage respectively between normal and obfuscated code.

From the tables,it is evident that the change in energy consumption ranges from 0% to 11.32% for different obfuscation techniques and that the maximum change is observed in basicmath when data obfuscation is applied. In contrast, minimum change is observed for matrix multiplication and Vernam cipher when data and lexical obfuscation are applied respectively. In the following, we analyze the energy

Table 2: Control Flow Obfuscation

App	%ΔP	%ΔT	%ΔE	%ΔS
Matrix	CF:1.27	CF:0	CF:1.27	CF:3.76
	DI:2.17	DI:4	DI:6.26	DI:8.85
	EL:2.35	EL:4	EL:6.45	EL:4.5
	LT:1.87	LT:4	LT:5.95	LT:9.01
Basicmath	CF:3.62	CF:5.88	CF:9.71	CF:5.57
	DI:0.95	DI:5.88	DI:6.89	DI:1.18
	EL:1.21	EL:5.88	EL:7.16	EL:1.58
	LT:0.38	LT:2.94	LT:3.33	LT:0.57
Bitcount	CF:1.18	CF:1.4	CF:2.6	CF:3.59
	DI:1.8	DI:1.41	DI:3.24	DI:3.83
	EL:1.37	EL:1.41	EL:1.37	EL:4.43
	LT:0.12	LT:1.41	LT:0.12	LT:1.02
Vernam	CF:0.5	CF:8.33	CF:8.88	CF:3.13
	DI:0.82	DI:8.33	DI:9.22	DI:2.99
	EL:1.01	EL:8.33	EL:9.43	EL:3.74
	LT:0.25	LT:8.33	LT:8.61	LT:2.51

CF:Control Flow Flattening DI:Deadcode Insertion
EL:Extended Loop Condition LT:Loop Transformation

and performance impact pertaining to each of the obfuscation techniques.

Lexical Obfuscation:

The least values for change in energy consumption across all applications indicate that lexical obfuscation minimally impacts power consumption. This is due to the negligible overhead involved in carrying out the operations such as changing variables, removing comments etc. Thus overall structure of the actual code is maintained with minimal changes.

Similarly, we observe that execution time between normal and obfuscated code remains similar for the four applications. Thus lexical obfuscation incurs minimal impact on performance.

Data Obfuscation:

In the case of data obfuscation, we observe a noticeable increase in energy consumption. In particular, data obfuscation of basicmath consumes higher energy compared to other transformations. This is due to the data manipulation operations such as cubic function solving, square root evaluation, degree to radian conversion etc that are contained in data obfuscation which incurs significant amount of computation resulting in high energy.

Similarly, we observe a maximum increase in execution time of 8.82 % for basicmath program followed by Vernam cipher of 4.16% and close to negligible for matrix multiplication and bitcount benchmarks. Thus data obfuscation negatively impacts performance of basicmath due to the compute intensive operations.

Control Flow Obfuscation:

Among the four transformations for control flow obfuscation, we observe maximum energy consumption for extended loop condition and loop transformations. This is primarily due to the increase in number of loop operations which may be computationally intensive and that they consume more energy compred to other transformations.

Similarly, we observe a noticeable increase in execution time for the basicmath and Vernam cipher benchmarks due to control obfuscation. This can be attributed to the increase in the number of data manipulation operations compared to other benchmark applications.

Combined Obfuscation

From the tables, it is evident that combined obfuscation incurs a maximum impact on energy consumption. In particular we observe a maximum increase in energy consumption of 13.89% for Vernam cipher followed by basicmath and bitcount benchmarks that are at 13.32% and 13.8% respectively. This is primarily due to the greater code size after combining the techniques of obfuscation which resulted in high energy consumption.

Similarly, combined obfuscation negatively impacts performance by increasing the execution time of obfuscated programs due to combined obfuscation.

5.1 Trace Analysis

To analyze the long-term impact of code obfuscation, we gather power traces using SMU for different obfuscation techniques such as lexical, data and control flow transformations. Figures 2, 3 and 4 pictorially represent the results for lexical, data and control flow transformations pertaining to each of the applications. From the figure, it is evident that for certain transformations such as lexical obfuscation, power traces are similar and variations are minimal. To examine the degree of correlation between traces, we utilize a statistical measure called Pearson correlation coefficient.

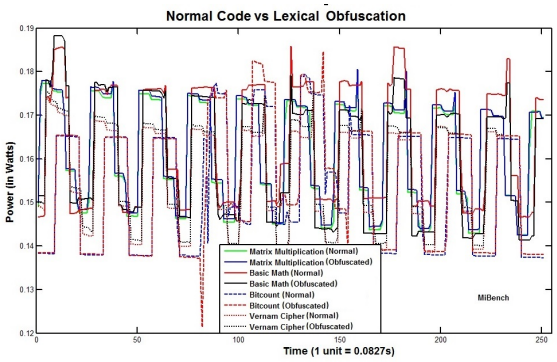

Figure 2: Lexical Obfuscation

Pearson correlation coefficient is a statistical measure used to estimate the degree of similarity between different kinds of data. A value of 1.00 implies perfect correlation, while 0 and -1 indicate no-correlation and negative correlation respectively. We estimate the correlation coefficient between obfuscated code and non-obfuscated code for the transformations pertaining to each of the applications. Table 3 contains the results for correlation.

From the table, we make the following inferences. Lexical obfuscation maintains the structure information of the actual code without changing the original assembly code. This explains the similarity in cross correlation values for normal and lexically obfuscated code. In the case of data obfuscation, it is evident from figure 3 that the power traces are not similar. This behavior is validated by the cross correlation values computed for individual power traces. Similarly, transformations that involve loop operations such as extended loop and loop transformations exhibit wide variations in control flow obfuscation. However, we observed a

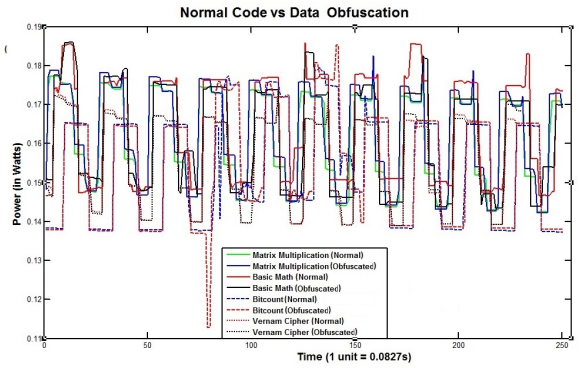

Figure 3: Data Obfuscation

maximum variation in traces in the case of combined obfuscation. This is further validated by the relatively least correlation coefficient for these transformations.

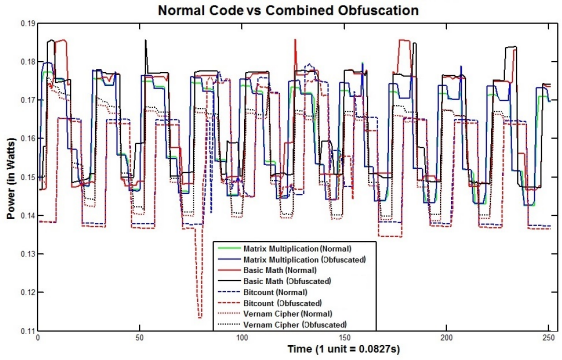

Figure 4: Combined Obfuscation

6. ENERGY OPTIMIZATION

Our study shows that obfuscation contributes significantly towards the energy consumed by embedded devices and that the rate of energy consumed depends on the kind of obfuscation. Thus, optimizing the energy consumed due to obfuscation is necessary so that systems meet a given power budget. As mobile applications are increasingly becoming critical and that the power consumed due to securing them may proportionately increase, minimizing energy use is critical considering the long-term sustainability of embedded devices.

We propose the following approaches for energy optimization based on the insights obtained from the energy analysis. The proposed approaches emphasize the need for lightweight security and analysis of energy-performance-security tradeoffs for embedded devices.

Application-aware Obfuscation:

The primary purpose of energy analysis of different techniques for code obfuscation is to develop an energy profile of different transformations depending on the embedded platform. Currently, open-source tools for code obfuscation such as tigress [8] are not designed for embedded devices. Thus one of the approaches is to port source code from tigress to embedded devices for energy profiling. Integrating energy profiles into tigress would enable developers to develop lightweight transformations depending on the needs of the applications and the overall system-level power budget.

Table 3: Pearson Correlation for Normal and Obfuscated Code

App	Lexical	Data	Control	Combined
Matrix	0.9977	0.9903	CF:0.9950 DI:0.9946 EL:0.9934 LT:0.9956	0.9925
BasicMath	0.9811	0.9528	CF:0.9005 DI:0.9827 EL:0.9947 LT:0.9812	0.8871
BitCount	0.9922	0.9445	CF:0.9874 DI:0.9571 EL:0.9909 LT:0.9914	0.9575
Vernam	0.9974	0.9866	CF:0.9955 DI:0.9806 EL:0.9903 LT:0.9948	0.9852

CF:Control Flow Flattening DI:Deadcode Insertion
EL:Extended Loop Condition LT:Loop Transformation

Obfuscation vs Encryption:

While obfuscation provides time-limited protection for embedded devices, encryption enhances security through periodic renewal of keys combined with multiple cryptographic algorithms. There are trade-offs associated with obfuscation and encryption for embedded devices and both depend on the size of the source code that needs to be obfuscated or encrypted. Although encryption can be part of obfuscation, we have considered them to be separate in our analysis. Our goal is to assess the energy benefits of obfuscation and encryption and analyze their applicability in embedded applications considering their energy profile and the effectivess of security. In addition, approaches for integrating obfuscation and encryption in an energy efficient manner can be explored.

Energy-Performance-Security trade-offs:

While energy and performance of algorithms for code obfuscation can be estimated, estimating security is of increasing importance towards analyzing the energy-performance-security trade-offs for embedded applications. However, estimating security is challenging due to the lack of available metrics. One of the approaches to estimate security is through approximation [26], where a regression-based model is constructed based on different security parameters pertaining to each of the obfuscation based techniques that are considered independent variables and the resulting security to be the dependent variable. While the approach presented in [26] shows promise, security for different obfuscation techniques need to be estimated and the resulting

energy-performance-security trade-offs analyzed.

7. CONCLUSION

In this work, we analyze the impact of code obfuscation on energy usage for embedded devices. Particularly, we model algorithms for code obfuscation such as Lexical, Data and Control obfuscation and measure the energy and performance of normal and obfuscated code using a measurement tool. Our analysis on a set of benchmarks reveals that while lexical obfuscation has a minimal impact on power consumption, data and control obfuscation was shown to have a significant impact in terms of performance and power consumption. The insights obtained from our analysis can be used for application-aware obfuscation, assessing the energy benefits between obfuscation and encryption and analyzing the energy-performance-security trade-offs.

8. REFERENCES

[1] Allatori java obfuscator. *http://www.allatori.com/*.

[2] Dasho - preemptive solutions. *http://www.preemptive.com/products/dasho*.

[3] Keil mdk-arm. *http://www2.keil.com/mdk5*.

[4] Keithley 2400 sourcemeter. *http://www.tek.com/keithley-source-measure-units/keithley-smu-2400-series-sourcemeter*.

[5] Zelix klassmaster. *http://www.zelix.com/klassmaster/*.

[6] S. Banescu, M. Ochoa, and A. Pretschner. A framework for measuring software obfuscation resilience against automated attacks. In *2015 IEEE/ACM 1st International Workshop on Software Protection*, pages 45–51, May 2015.

[7] S. Chow, Y. Gu, H. Johnson, and V. A. Zakharov. An approach to the obfuscation of control-flow of sequential computer programs. In *Proceedings of the 4th International Conference on Information Security*, ISC '01, pages 144–155, London, UK, UK, 2001. Springer-Verlag.

[8] C. Collberg, S. Martin, J. Myers, and B. Zimmerman. The tigress diversifying c virtualizer. *http://tigress.cs.arizona.edu*.

[9] C. Collberg, C. Thomborson, and D. Low. A taxonomy of obfuscating transformations. Technical report, Department of Computer Science, The University of Auckland, New Zealand, 1997.

[10] S. Drape. Generalising the array split obfuscation. *Information Sciences*, 177(1):202–219, 2007.

[11] M. Dukovic and E. Varga. Load profile-based efficiency metrics for code obfuscators. *Acta Polytechnica Hungarica*, 12(5), 2015.

[12] D. Economou, S. Rivoire, and C. Kozyrakis. Full-system power analysis and modeling for server environments. In *In Workshop on Modeling Benchmarking and Simulation (MOBS*, 2006.

[13] X. Fan, W.-D. Weber, and L. A. Barroso. Power provisioning for a warehouse-sized computer. In *Proceedings of the 34th Annual International Symposium on Computer Architecture*, ISCA '07, pages 13–23, New York, NY, USA, 2007. ACM.

[14] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. Mibench: A free, commercially representative embedded benchmark suite. In *Proceedings of the Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop*, WWC '01, pages 3–14, Washington, DC, USA, 2001. IEEE Computer Society.

[15] C. Isci and M. Martonosi. Runtime power monitoring in high-end processors: Methodology and empirical data. Technical report, Princeton University Electrical Eng. Dept., September 2003.

[16] M. Kainth, L. Krishnan, C. Narayana, S. G. Virupaksha, and R. Tessier. Hardware-assisted code obfuscation for fpga soft microprocessors. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, DATE '15, pages 127–132, San Jose, CA, USA, 2015. EDA Consortium.

[17] S. Khan, P. Xekalakis, J. Cavazos, and M. Cintra. Using predictive modeling for cross-program design space exploration in multicore systems. In *Proceedings of the 16th International Conference on Parallel Architecture and Compilation Techniques*, PACT '07, pages 327–338, Washington, DC, USA, 2007. IEEE Computer Society.

[18] E. Lafortune et al. Proguard. *http://proguard.sourceforge.net*, 2004.

[19] S. Microelectronics. Stm32f0discovery. *STM32F0 highperformance discovery board, User manual*, 2013.

[20] S. Praveen and P. S. Lal. Array data transformation for source code obfuscation. *Performance Improvement*, 658:6515, 2007.

[21] C. Sahin, P. Tornquist, R. Mckenna, Z. Pearson, and J. Clause. How does code obfuscation impact energy usage? In *Proceedings of the 2014 IEEE International Conference on Software Maintenance and Evolution*, ICSME '14, pages 131–140, Washington, DC, USA, 2014. IEEE Computer Society.

[22] K. Sangaiah, M. Hempstead, and B. Taskin. Uncore rpd: Rapid design space exploration of the uncore via regression modeling. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '15, pages 365–372, Piscataway, NJ, USA, 2015. IEEE Press.

[23] S. Sankaran. Predictive modeling based power estimation for embedded multicore systems. In *Proceedings of the ACM International Conference on Computing Frontiers*, CF '16, pages 370–375, New York, NY, USA, 2016. ACM.

[24] T. Toyofuku, T. Tabata, and K. Sakurai. Program obfuscation scheme using random numbers to complicate control flow. In *International Conference on Embedded and Ubiquitous Computing*, pages 916–925. Springer, 2005.

[25] S. Wang, H. Chen, and W. Shi. Span: A software power analyzer for multicore computer systems. *Elsevier Sustainable Computing: Informatics and Systems*, page In press, 2011.

[26] Y. Wu, H. Fang, S. Wang, and Z. Qi. A framework for measuring the security of obfuscated software. In *Proceedings of the 2010 International Conference on Test and Measurement*, ICTM '10, 2010.