2012 Third International Conference on Emerging Security Technologies

Investigation of Properties of ICmetrics Features

Yevgeniya Kovalchuk, Huosheng Hu,
Dongbing Gu, Klaus McDonald-Maier
School of Computer Science & Electronic Engineering
University of Essex
Colchester, UK
yvkova@essex.ac.uk; hhu@essex.ac.uk;
dgu@essex.ac.uk; kdm@essex.ac.uk

Daniel Newman, Steve Kelly, Gareth Howells
School of Engineering and Digital Arts
University of Kent
Canterburry, UK
dwn3@kent.ac.uk, S.W.Kelly@kent.ac.uk
W.G.J.Howells@kent.ac.uk

***Abstract*—The ICmetrics technology is concerned with identifying acceptable features in an electronic system's operation for encryption purposes. Ideally, the nature of the features should be identical for all of the systems considered, while the values of these features should allow for unique identification of each of the systems. This paper looks at the properties of the Program Counter of a processor core as a potential ICmetrics feature, and explores how the number of its samples being inputted into the ICmetrics system affects stability of the system's performance.**

ICmetrics; security; encryption; embedded systems; autonomous systems

I. INTRODUCTION

As the modern society fully relies on operation of digital devices, the importance of developing a security infrastructure for digital data storage and transmission is essential. As an alternative to traditional ways to secure data (e.g. passwords, biometrics, etc.), the ICmetrics technology (Integrated Circuit metrics) generates encryption keys directly from measurements taken from electronic devices. This could be advantageous in many real world scenarios where human intervention is not possible, limited or not desirable. For instance, an ICmetric system is currently under development as part of the SYSSIAS project (Autonomous and Intelligent Healthcare System), which is aimed at developing an intelligent electrically powered wheelchair for patients with various disabilities.

A theoretical description of the operation of the ICmetrics technology can be found in our earlier work [1]. In summary, the ICmetrics system is a two phase system. In the first phase, a number of known electronic devices are used as a calibration set and desired features associated with the devices are measured. Based on the frequencies of the feature occurrences, the feature distributions are generated, first within the given value scale for each sample device, and then normalised across all devices and features employed. The output of the first stage is a set of normalisation maps containing normalised feature distributions for each device.

The second, operational, phase of the ICmetrics system is applied each time an encryption key is required for a given device. After measuring feature values and applying the normalisation map as created in the first phase for the device, a unique basic number is generated using a suitable technique for combining the features. An encryption key can be further derived from this basic number.

The main requirement for the basic number is that it should remain constant for a given device on each attempt of its generation, but distinct from the basic numbers generated for other devices employed in the operational set. Furthermore, it should not be possible to derive or estimate the encryption keys generated for other devices based on the basic number of a given device. In order to achieve this, it is important to find such features associated with electronic devices, which allow for separation of the devices in the feature space.

In this paper, we explore the Program Counter of a processor core as a potential ICmetrics feature and test if it could be used to determine a device uniquely. By taking only one feature, we want to evaluate whether it is suitable for ICmetrics purposes, and how manipulating it affects the ICmetrics system performance. In particular, we look at how changing the employed number of the Program Counter values as recorded during a device operation (referred latter as samples) influences the system's ability for device separation.

For the purposes of this study, we have developed and built an embedded system and loaded a number of software applications on it so as to simulate behaviour of several electronic devices. We have used two different tracing methods in order to record the Program Counter values. In addition to the characteristics of the Program Counter as a potential ICmetrics feature itself, we also explore how each of the proposed tracing methods affects the process of the device separation in the feature space. The device separation is an important stage as results obtained during this phase determine the strength of the encryption key that can be generated further.

In order to evaluate the Program Counter as a potential ICmetrics feature and the tracing methods to obtain the feature values, we have developed a system that attempts to separate devices in the feature space. More specifically, the system takes the feature values for all devices employed as input, generates normalisation maps for each of the devices based on these values [1], and plots normalised feature distributions for all of the devices in one space. The successful output of the system is observed when all devices

978-0-7695-4791-6/12 $26.00 © 2012 IEEE
DOI 10.1109/EST.2012.22

can be uniquely identified and there are no overlaps in the distributions plotted in one space. This is the main criterion against which we evaluate the Program Counter as a potential ICmetrics feature; if we could achieve the separation of the devices based on the Program Counter values for each of the devices, then we can say that this is a suitable feature.

The rest of the paper is organised as follows. First, we describe our hardware and software platform which we have used to extract feature values. Then, we explain the analysis of the data we have performed and provide interpretation of the results. Finally, we summarise the paper with some suggestions for future work.

II. Experimental Setup

Building an experimental platform for extracting ICmetrics features involves several stages: (i) designing the hardware-software test-bench; (ii) programming simulations of embedded systems' operation; (iii) developing tracing methods for data acquisition; (iv) recording feature values for their further analysis as required by ICmetrics research. The following subsections describe implementation of these stages in turn.

A. Hardware-software test-bench

The experimental platform for extracting ICmetrics features requires a hardware board for hosting an embedded system, and also a soft- and hardware infrastructure for loading programs onto the board, as well as tracing programs' execution in real time. For this research, we have employed:

- a low resource embedded system based on Atmel AT91SAM7S256 microcontroller [2] and 64Kbytes SRAM memory;
- JTAG programming port for programming the microcontroller;
- Open On-Chip Debugger (OOCD) [3] in conjunction with the JTAG programming port to trace program execution;
- Eclipse [4] as a software environment to interface with the hardware platform (i.e., to develop, compile, and load software code onto the board);
- a logging program that we have developed specifically for this study; it connects to the board via a telnet port and logs required features.

B. Simulation of system operation

To simulate performance of several devices, we have employed a number of algorithms from the automotive package of the MiBench suite of benchmark algorithms [5], referred later as "devices". At this early stage of our research, we have employed basic low complexity software routines, namely:

- angle conversion (Device 1);
- bit count (Device 2);
- cubic function (Device 3);
- square roots (Device 5).

In addition, we have included a program generating random numbers to simulate Device 4.

We have loaded each of the programs in turn onto the board (section *II.A*), traced their execution (using the methods outlined in section *II.C*), and logged the Program Counter values (as described in section *II.D*).

C. Tracing methods

We have employed two tracing methods to log the Program Counter values which we used as ICmetrics features. The first one is the single stepping method to obtain benchmark data against which to evaluate the second method, which is sampling. Essentially, both methods halt the CPU by issuing OOCD commands [3] via a telnet port, and register the Program Counter values. Note that while these methods affect program execution times, they do not change the execution flow.

The difference between the two methods lies in the frequency and completeness of obtaining data. The single stepping tracing method logs every single CPU instruction, while the sampling method does this only at regular intervals, at the rate of 50Hz in this case. We have chosen this sampling rate since it is the highest throughput that our JTAG programming port supports and we aimed at testing a fast logging method to be practical in real time applications. These settings mean that the single stepping method provides the complete profile of the program execution; however it is very slow. The sampling method on the other hand allows speeding up logging considerably (thus, affecting the systems normal executing much less); however since it does not log the Program Counter values at every step, significant parts of the program execution profile may not be identified.

The paper is concentrated on the later problem of estimating the effect of such a limited knowledge of the execution profile on the efficiency of the ICmetrics system.

D. Data acquisition

In this study, we have explored only the Program Counter as an ICmetrics feature. Taking only one feature allows for a controlled evaluation of its suitability for the ICmetrics system. We expect the Program Counter to be a suitable feature since the set of its distinct values is finite and is the same for a certain device (assuming the full program profile is taken), but is likely to vary from one device to another. Having the requirement for ICmetrics features that they should allow for separation of the considered devices in the feature space, we test if this is the case with the Program Counter.

To obtain feature data, we have logged the Program Counter values while running each of the programs (section *II.B*) using each of the two tracing methods (section *II.C*) in turn.

III. Data Analysis

As described in section II, we have logged the Program Counter values while running a number of programs on our hardware platform (this way simulating the behaviour of

several electronic devices), using two different tracing methods. Based on the logs, we have found that there are certain particularities related to the nature of the Program Counter as a feature which need to be addressed first before attempting to generate normalised feature distributions. The following sections describe the peculiar characteristics of the feature and how we dealt with them.

For further discussion, it is useful to note the difference between distinct feature values (unique values of the Program Counter) and total number of values (referred later as samples) recorded in the devices' logs. This is because a program running on a device can use the same address several times during its operation. In this study, we manipulate only with samples (i.e. all feature values as recorded in the logs) regardless the number of unique addresses that they span over. However for the future, it would be interesting to investigate how the ratio between the unique number of values and total number of samples recorded in the logs determines the ability of our system to separate the devices.

A. Feature Distributions

In order to compare how the devices use the memory (feature space in our case), we have plotted frequencies of all Program Counter values logged using the single stepping tracing method for all devices in the address-frequency domain. As can be seen from Figure 1, there are certain areas in the address space that are clearly occupied by one device only, while certain addresses are used by several devices. In other words, there are overlaps in the devices' profiles. It must be noted however, that the intensity (or frequency) of using the shared addresses is different for different devices.

As opposed to Figure 1 where all feature values are depicted, Figure 2 plots only unique addresses for each device, i.e. the addresses that are used by a given device, but not by the rest of them. It can be noticed from Figure 2 that there are no unique values of the Program Counter for device 2. However, is still difficult to say if the Program Counter allows separation of the devices in the feature space (which is the requirement for suitable ICmetrics features). In particular, since other devices contain several Program Counter values that are unique to them, it might be possible to take a portion of these values and attribute them to device 2, taking them out from the profile(s) of the device(s) that contained these values. That way, each device will have the Program Counter values that are unique to them.

In order to test if such reassigning of feature values would allow for device separation in the Program Counter feature space, we have developed a mapping system that groups feature values for each device and builds normalised feature distribution maps based on the original frequencies of the feature occurrences. The mapping process has been implemented as described in our earlier work [6]; namely, the original feature distributions have been mapped into virtual Gaussian distributions for all considered devices placed next to each other.

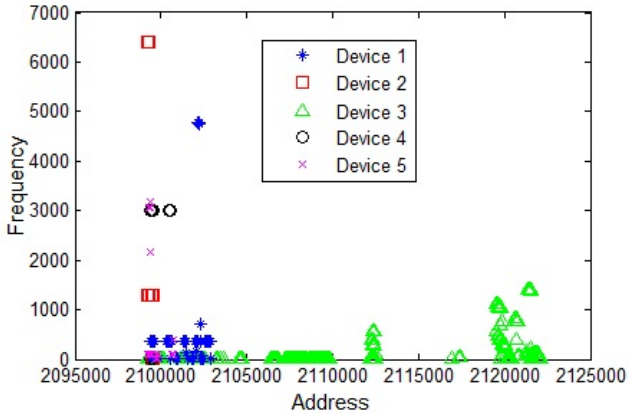

Figure 1. Frequencies of all feature occurrences.

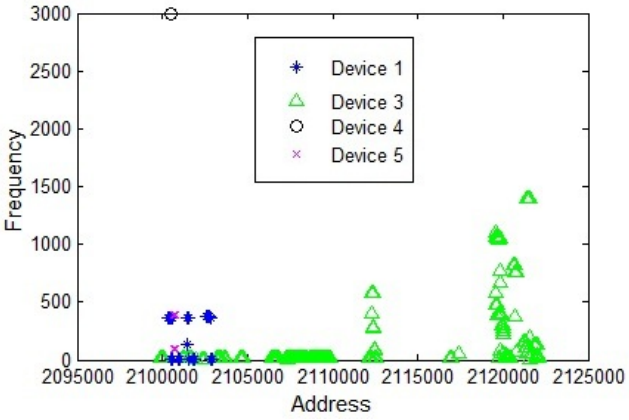

Figure 2. Frequencies of unique feature occurrences.

It should be noted that for this study, the devices have been mapped in order from device 1 to device 5. This means that any shared feature values were allocated to the device that has been mapped before the rest of the devices that had the same feature value (although more likely with another frequency). In the future however, we plan to improve the mapping algorithm by introducing a better method for dealing with shared feature values based on the frequencies of their occurrences.

B. Data offsetting

To render data suitable for the use within our mapping system, we have first performed its pre-processing as follows. From analysis of the Program Counter logs, it can be stated that the Program Counter varies in address values across the devices. However, each one of these values lies between 2,000,000 and 2,200,000. Note that this interval of feature values is specific for the combination of our ARM processor and 64Kbyte memory (see section *II.A*) and may be different if another hardware platform is employed. At the same time, the range of the Program Counter values is always fixed and is determined by the size of the memory

included into the hardware system. The number of possible feature values is also restricted by the fact that the Program Counter takes only even address values. Therefore, we can conclude that in our case the maximum number of distinct feature values for any of the devices could not exceed 100,000.

Taking into consideration our observations on the possible range of the feature values, we have offset the data to make it suitable for further processing. In particular we have extracted the minimum address value present in the whole range of values, using the following formula:

$$Offset\ Value = (Program\ Counter - Program\ Counter_{(min)}) + 1$$

where $Program\ Counter_{(min)}$ is equal to 2,000,000 based on the configuration considered in this paper.

Examples of the resulting feature values are provided in Table I. Such offsetting has not only helped in reducing the memory required for data processing and manipulation, but also made the devices more sensitive to any changes in data values.

TABLE I. THE PROGRAM COUNTER OFFSETTING

Program Counter	Offset Value
2000000	1
2000100	101
2000300	301
2000600	601

C. Device separation

Using the system that produces the normalised feature distribution maps (see section *III.A*) over the offset data (section *III.B*), we have established that our devices can be separated in the Program Counter feature space (Figures 3-6). Based on our earlier discussion of the criteria for suitable ICmetrics features (see Introduction), we can conclude that the Program Counter is a suitable feature.

However as it was demonstrated on Figure 2, device 2 does not contain any unique Program Counter values as compared to the rest of the devices. Although it is still possible to uniquely identify the devices in our case (since the rest four devices do contain values that are unique to them), we must admit that enrolling programs different from the ones we used in this study may result in different mapping values, even to the extend when several devices overlap in the Program Counter feature space.

On the other hand, analysis of the logs allows us to suggest that the Program Counter may still serve as a potential source for extracting ICmetrics features. In particular, instead of taking its raw values, the features can be derived from the Program Counter frequency or sequence domain. In other words, while several programs may use the same address space, the frequency and sequence of the Program Counter values in the execution flow may be different. We intend to investigate these potential sources of data in our future work.

Furthermore, we have found one more condition that influences the ability of the Program Counter to separate devices. Namely, this condition is related to the number of samples (feature values) employed for generating normalisation maps, and is detailed below.

When analysing the Program Counter logs, we have noticed that the number of the logged Program Counter values (samples) varies from one device to another (see Table II). This number depends not only on the program each device is running, but also on the tracing method employed to log the feature values.

Since the processing times of the mapping system directly correlates with the number of feature values that are input into it (i.e., the more feature values are taken, the longer it takes to generate distribution maps), we want to explore how mapping results are affected by varying the input number of the Program Counter values. We are also interested to check if taking this number aligned to the minimum number of samples recorded across the devices (device 5 in our case) would still allow for device separation in the feature space, while reducing processing times considerably (as compared to aligning the input number to the maximum number of samples recorded across the devices).

In order to see how the number of samples being inputted into the system affects the mapping results, we have mapped the five devices based on the following four sets of data:

1) Data set that includes the same (limited) number of feature values from the Program Counter logs recorded by the single stepping tracing method (Figure 3). In particular, we have taken the lowest number of samples (102,132 for the device 5 as in Table II) and used that number across all devices.
2) Data set that includes the same limited number of samples as above, but based on the logs recorded with the sampling tracing method, meaning a reduced number of distinct feature values as compared to the data set 1 (Figure 4).
3) Data set that includes all data recorded for each device with the single stepping tracing method, i.e. without limiting the number of samples being inputted into the mapping system (Figure 5).
4) Same as data set 3 but based on the logs recorded with the sampling tracing method (Figure 6).

TABLE II. NUMBER OF PROGRAM COUNTER VALUES (SAMPLES) RECORDED FOR EACH DEVICE

Device	Single stepping logs	Sampling logs
1	263,565	754,891
2	104,972	191,354
3	205,482	306,673
4	138,012	198,787
5	102,132	273,655

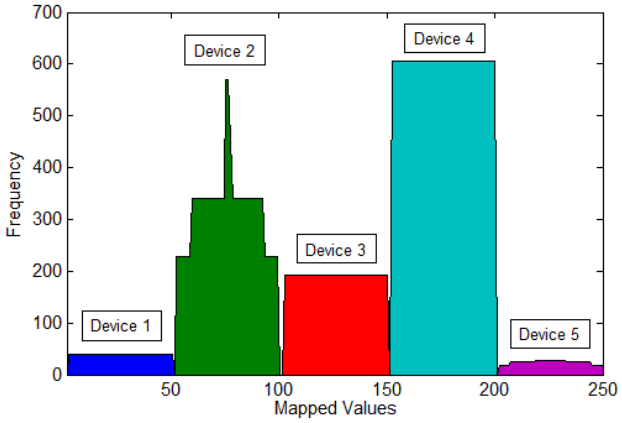

Figure 3. Mapped feature values obtained based on a limited set of samples from the single stepping logs.

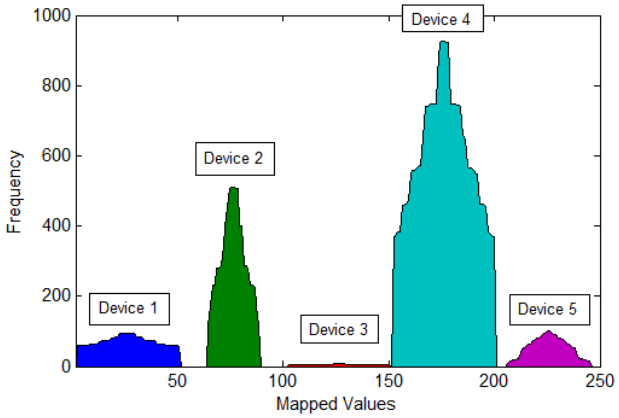

Figure 4. Mapped feature values obtained based on a limited set of samples from the sampling logs.

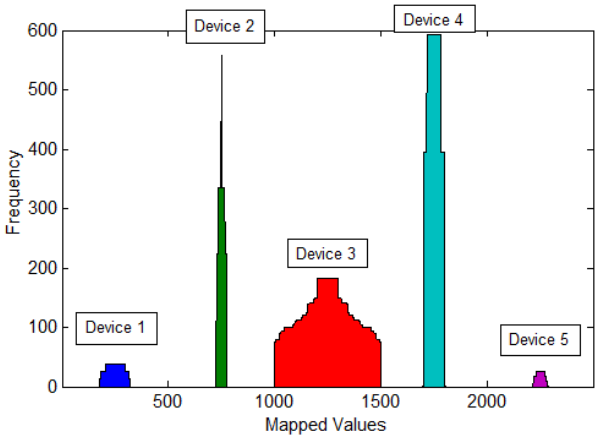

Figure 5. Mapped feature values obtained based on the single stepping logs with no samples limit.

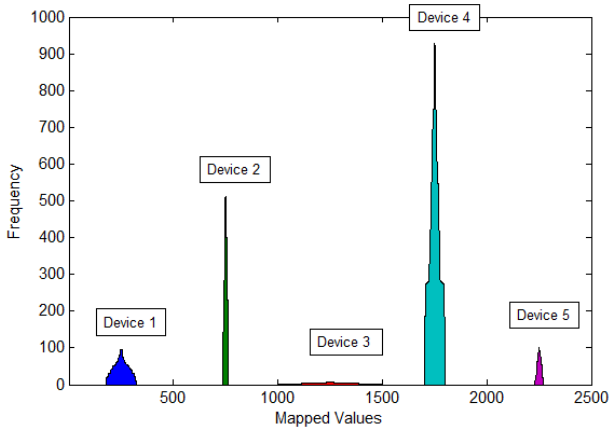

Figure 6. Mapped feature values obtained based on the sampling logs with no samples limit.

It can be noticed from Figures 3-6 that by taking less samples (i.e. the Program Counter values) than have been logged, the mapped feature distribution curves become more block-like (Figures 3 and 4) as compared to more smooth curves if all data values are considered (Figures 5 and 6). This is particularly true for the devices 3, which has a smooth shape of the mapped feature distribution curve in Figure 5, where there is no limit to the number of samples considered, and a square shape on Figures 3, where this limit is introduced.

Note that on Figures 4 and 6, where mapped feature distributions are plotted based on the data obtained by the sampling tracing method, the device 3 seem to disappear completely. This means that not enough distinct Program Counter values are present in sampling logs in order to identify the device.

By comparing Figures 3 and 4 with 5 and 6, it can also be noticed that the number of samples included into the system affects not only the shape of the mapped feature distribution curves (and their presence), but also the mapped feature values. While the same devices are used, the mapped values are not in the same place (see the mapped values marked on axis X on Figures 3 and 4 as opposed to these values on Figures 5 and 6). The larger values on Figures 5 and 6, and the empty spaces between the devices reflect the fact that while all logged Program Counter values has been allowed to be input into the mapping system, the considered devices had this number different. For example, device 3 has more unique values compared to the rest of the devices (which can be clearly seen from Figure 2), which is why mapped space allowed for each device has been aligned to this device. Since the rest of the devices had less unique values, they were unable to fill in the allowed space completely.

These observations allow us to conclude that any drop in the number of samples included into the system (as compared to a full device's profile) would change the mapped values, which in its turn would impact encryption keys derived from these values. In our future work, we intend to explore techniques for finding an optimal number of samples and their distinct values to include into the system so as to speed up the logging and mapping processes on the one hand (to be useful in real time applications), and to achieve generation of strong encryption keys on the other hand.

IV. Conclusion

This paper has explored the properties of the Program Counter as an ICmetrics feature and investigated how the number of samples of the feature values being employed affects the ICmetrics system's performance.

The main criterion for ICmetrics features is that they should allow for unique identification of the considered devices. In other words, it should be possible to separate the devices in the feature space based on the observed feature values associated with these devices.

We have shown that while taking separate Program Counter values may not always allow to satisfy this criterion, the Program Counter may still serve as a suitable source for extracting ICmetrics features. In particular, the features may be derived from frequencies of the Program Counter occurrences and their sequences observed during programs' execution flow [7]. Furthermore, features may be obtained from other sources (in addition to the Program Counter) related to devices' operation. In our future work, we plan to investigate alternative sources for ICmetrics features and build a system for separating devices in a multi-dimensional space.

In this stud, we have also found that in order to activate the separating ability of the Program Counter as a feature, it is important to correctly identify the number of samples and their distinct values to include into the system that separates devices. If taking not enough feature values, there is a chance that a device will not be identified. However, logging all feature values may not be possible in real time applications. Moreover, different number of employed features provides different mapped feature values. These values will eventually be used to generate encryption keys, which is why it is important to investigate the optimal number of samples to employ so as to minimise processing times and maximise the strength of the keys at the same time. We plan to address this issue in our future work.

Acknowledgment

The authors gratefully acknowledge the support of the UK Engineering and Physical Sciences Research Council under grant EP/K004638/1 and the EU Interreg IV A 2 Mers Seas Zeeën Cross-border Cooperation Programme – SYSIASS project: Autonomous and Intelligent Healthcare System (project's website http://www.sysiass.eu/).

References

[1] Y. Kovalchuk, G. Howells, and K.D. McDonald-Maier, "Overview of ICmetrics Technology – Security Infrastructure for Autonomous and Intelligent Healthcare System", International Journal of u- and e-Service, Science and Technology, vol. 4, no. 3, Sep. 2011, pp. 49-60.

[2] Atmel's SAM7S datasheet: http://www.atmel.com/Images/doc6175.pdf

[3] Online OpenOCD User's Guide: http://openocd.sourceforge.net/doc/html/index.html#Top

[4] Eclipse official web-site: http://www.eclipse.org/

[5] 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", Proceedings of the International Workshop on Workload Characterization, 2001, pp. 3-14.

[6] E. Papoutsis, G. Howells, A.B.T. Hopkins, K.D. McDonald-Maier, "Integrating Multi-Modal Circuit Features within an Efficient Encryption System", Journal of Information Assurance and Security, vol. 2, no. 2, 2007, pp. 117-126.

[7] Y. Kovalchuk, W.G.J. Howells, H. Hu, D. Gu, K.D. McDonald-Maier, "ICmetrics for Low Resource Embedded Systems", Proceedings of the third International Conference on Emerging Security Technologies, 2012.