Investigation of Properties of ICmetrics Features

Yevgeniya Kovalchuk[1], Huosheng Hu,
Dongbing Gu, Klaus McDonald-Maier[2]
School of Computer Science and Electronic Engineering, University of Essex
[1]*yvkova@essex.ac.uk*,
[2]*kdm@essex.ac.uk*

Daniel Newman, Steve Kelly,
Gareth Howells
School of Engineering and Digital Arts, University of Kent
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. 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 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.

1. 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, we are currently developing an ICmetrics system 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 is that it should be the same for the same device on each attempt of its generation, but always different from the basic numbers generated for other devices employed in the operational set. Furthermore, it should not be possible to derive 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 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 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 used to extract feature values. Then we explain the analysis of the data we performed and provide interpretation of the results. Finally, we summarise the paper with some suggestions for future work.

2. 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.

2.1. 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:

- ARM main board with Atmel AT91SAM7S256 microcontroller and 64Kbytes SRAM memory;
- JTAG programming port for direct access and control to various processor features (e.g., memory and internal registers) and external control of the processor (loading, executing, and debugging programs);
- Open On-Chip Debugger (OOCD) to trace program execution;
- Eclipse as an interface environment to develop, compile, load, debug, and trace software code on to the hardware platform;
- Software program that connects to the board via the telnet port and logs required features.

2.2. 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 [2], referred later as "devices", 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 2.1), traced their execution (using the methods outlined in section 2.3.), and logged the Program Counter (as described in section 2.4.).

2.3. 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. Both methods are non-intrusive, meaning they do not affect the residence of the programs in the address space.

Essentially, both methods halt the CPU by issuing OOCD commands, via the telnet port [3], and register the Program Counter values. The difference between the methods lies in the frequency and completeness of obtaining data. The single step tracing method logs every single CPU instruction, while the sampling method does this only at regular intervals, at the rate of 50MHz in this case. We have chosen this sampling rate since it is the highest frequency that our hardware supports and our aim is to trace as fast as possible to be practical in real time applications.

These settings mean that the single step method provides the complete profile of the program execution, while the sampling method only its approximation (the more samples are taken the closer the profile is to the complete one). Whilst the single step method is preferred to gain a full profile, it is very slow and not always practical to implement. The sampling method on the other hand allows speeding up logging considerably; however it does mean that not all data can be logged. The paper is concentrated on the later problem of estimating the effect of the loss in data on the efficiency of the ICmetrics system.

2.4. 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 while running each of the programs (section 2.2) using each of the two tracing methods (section 2.3) in turn. The size of the programs was adjusted using an oscilloscope in order to achieve practical logging times and comparable profiles for all of them. On the coding level, adjustment was implemented by tuning parameters of the loops involved in the programs.

3. Data analysis

As described in section 2, 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 due to 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.

3.1. Data offsetting

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 2.1) 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:

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

where Program Counter$_{(min)}$ is equal to 2,000,000.

Examples of the resulted feature values are provided in Table 1. 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 1. Feature 1: Program Counter offsetting

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

3.2. Device separation

Using the system that produces the normalised feature distribution maps [1], we have established that our devices can be separated in the Program Counter feature space (see Figures 3-5). 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, there is a certain 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 noticed that the number of the logged Program Counter values (samples) varies from one device to another (see Table 2). This number depends not only on the program each device is running, but also on the tracing method employed. However, the procedure of generating normalisation maps requires that the data are unified. To achieve this, the lowest number of samples (102,132 for the device 5 as in Table 2) has been taken and used across all the data sets. This has not only helped speed up the processing times, but also shows how the performance of the ICmetrics system is affected by taking a reduced set of feature values, as compared to the complete set of values registered in the devices' logs.

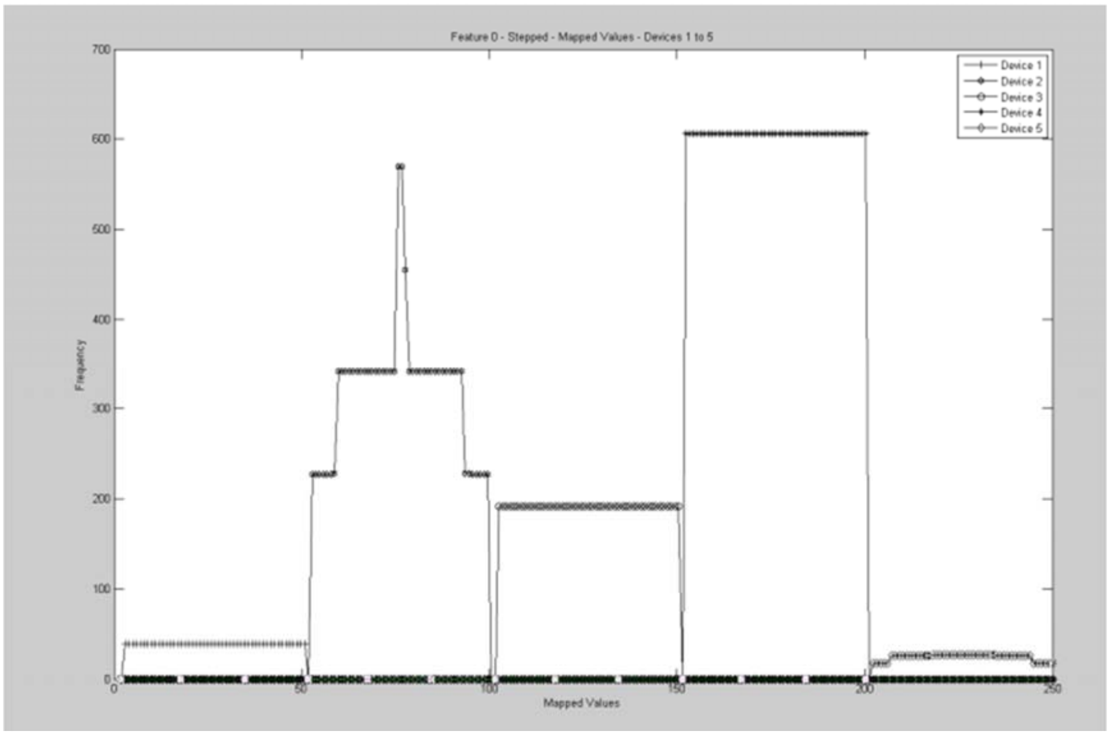

Figure 1. Mapped feature values obtained with the single stepping method and a limited set of samples

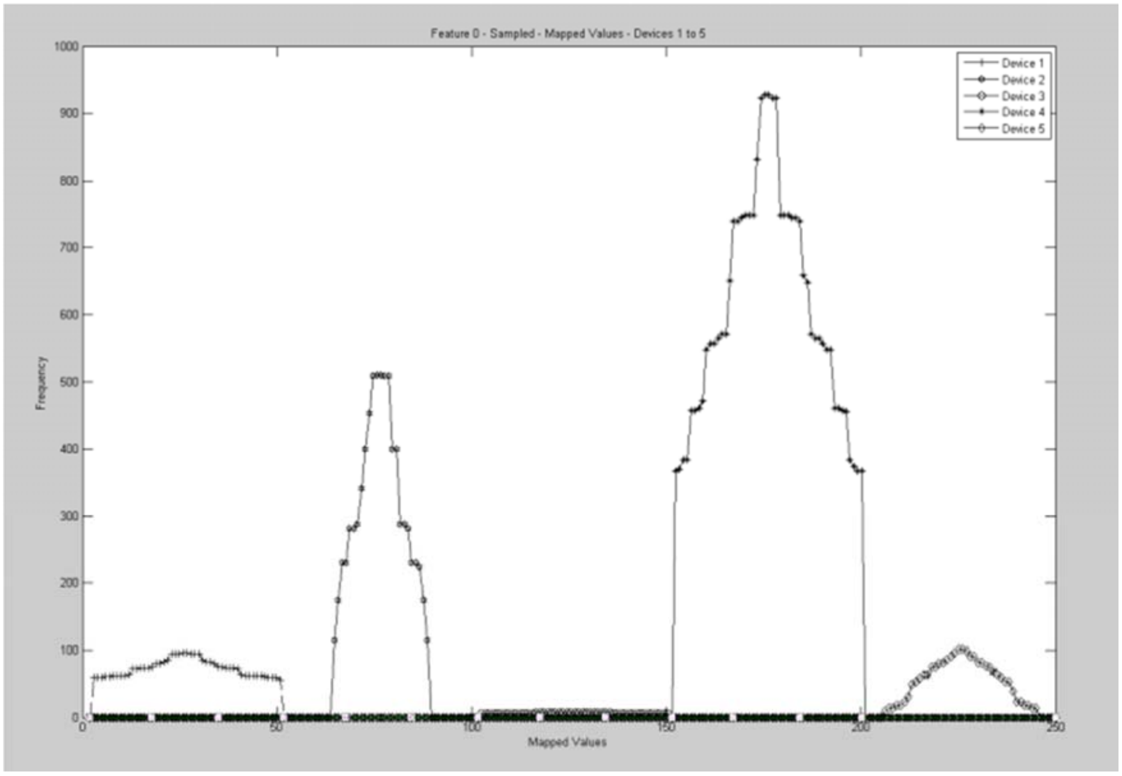

Figure 2. Mapped feature values obtained with the sampling method and a limited set of samples

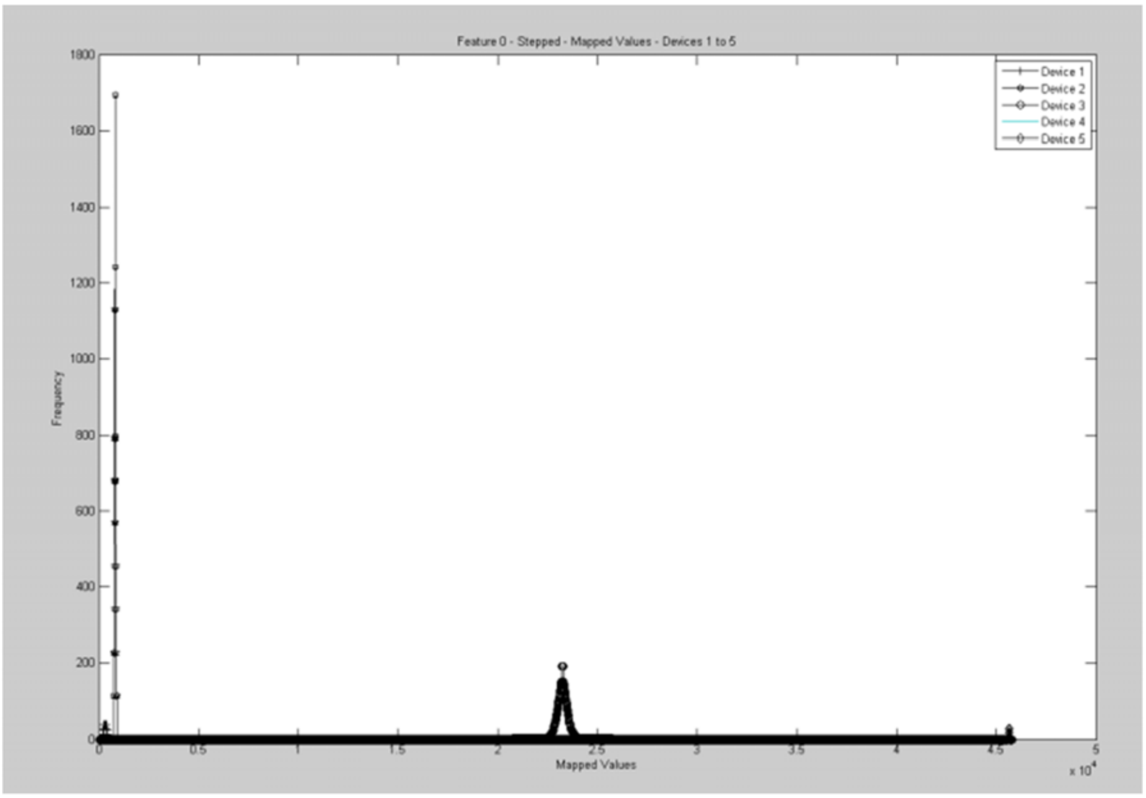

Figure 3. Mapped feature values obtained with the single stepping method and the whole set of samples

Table 2. Amount of samples each device has produced for the Program Counter feature

Device	Stepped Program Counter	Sampled Program Counter
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

In particular, we have found that by reducing the resolution of each device (in other words, by taking less samples, or the Program Counter values, than has been logged), they become more block-like as compared to a more smooth curves if all data values are considered. This is demonstrated with the devices 3, which has a smooth shape of the normalised 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 Figure 4, where normalised feature distributions are plotted based on the data obtained by the sampling tracing method (meaning the reduced number of samples), the device 3 seem to disappear completely. This means that not enough feature values (samples) has been taken in order to identify the device.

By comparing Figures 3, 4, and 5, it can also be noticed that the number of samples included into the system (with the largest on Figure 5 and the smallest on Figure 4) affects not only the shape of the normalised 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 throughout the graphs (see the modal values for each device on axis X). This means 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 to include into the system so as to speed up the logging process on the one hand (to be useful in real time applications), and to achieve generation of strong encryption keys on the other hand.

4. 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.

In this paper, we have shown that the Program Counter satisfies this criterion and therefore can be considered as a suitable ICmetrics feature. However, we have also found that in order to activate the separating ability of the Program Counter, it is important to correctly identify the number of samples to include into the system. If taking not enough feature values, there is a chance that a device will not be identified. Also, different number of employed features provides different mapped feature values (as we have shown with the single stepping and sampling tracing methods). 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 maximise the strength of the keys.

In addition to developing techniques for finding the optimal number of samples, we also want to explore how the ratio between the number of unique feature values compared to the total number of samples observed affects the system's performance.

Finally, it is worth noting that in this study the reading of feature values has been done without any internal code interfering with the actual system, but has been logged externally. From the point of view of tracing real time systems, our sampling method is preferred to the single stepping method, as the latter would take too long to retrieve the feature values.

5. 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*, 4 (3) (2011), 49-60.

[2] 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.

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

6. Acknowledgment

This research is supported at the School of Computer Science and Electrical Engineering at the University of Essex and at the School of Engineering and Digital Arts at University of Kent through the European Union ERDF Interreg IVA initiative under the SYSIASS grant.