Phase 2: True Random Generation on the APQ8060 Dragonboard

Kevin Burns
Department of Electrical and Computer Engineering
Virginia Tech
Blacksburg, Virginia
Email: kevinpb@vt.edu

Rob Lyerly
Department of Electrical and Computer Engineering
Virginia Tech
Blacksburg, Virginia
Email: rlyerly@vt.edu

Reese Moore
Department of Electrical and Computer Engineering
Virginia Tech
Blacksburg, Virginia
Email: ram@vt.edu

Philip Kobezak
Department of Electrical and Computer Engineering
Virginia Tech
Blacksburg, Virginia
Email: pkobezak@vt.edu

***Abstract*—With the mobile platform becoming more ubiquitous in modern society there is a greater need for the ability to run cryptographic algorithms on low power architectures. Most of these algorithms require a source of randomness. For the Android version of the Linux kernel this is provided by user input and timing data. It have been observed that this methodology does not provide the user with enough randomness and has caused either system performance issues or none-unique cryptographic keys. Therefore there is a high demand for a quick and power efficient random number generator for mobile devices.**

I. Introduction

A. Related Work and Background

In the Linux kernel, including the Android kernel, there is an entropy pool. Access to this pool is done through the random device located in /dev/random. If the pool is drained to below 128 bits, then the application reading from the pool will spin until more entropy is added to the pool. There also exists a urandom device which provides "unlimited" entropy. This device uses the same approach as the random device until the 128 bit barrier. At which point the urandom device does not block, but forgoes true randomness and is therefore vulnerable to cryptographic attacks.

In the Google Play Store there exists and application entitled Seeder. The developers of seeder are trying to speed up the Android experience by removing the need to spin on the random device. Their solution is when the random device's 128 bit limit is reached to pipe the false entropy from the urandom device back into the entropy pool. The developers of Seeder claim this brings the user noticeable speedups in day to day use of the Android Operating System. As this application has been downloaded over ten thousand times in the past 30 days, it is clear that this is an area of interest to the Android user community.

The task of finding true randomness on mobile platforms has been targeted before. In [5], the two main sources analyzed in this paper were the camera and microphone. The authors of this paper were unable to generate definite numbers for random number generation but were able to verify the validity. There were several advantages and disadvantages presented in this paper in regards to the camera as a random number generator. The advantages included the ability to produce considerable entropy even at $5\,^{\circ}\mathrm{C}$ and with the camera guard closed. However there were also some vulnerabilities, an attacker could use a halogen light bulb to over saturate the CCD to produce only 255 values for R, G, and B. This would highly influence the randomness given by the phone. The microphone's noise as a source of randomness was found to be 3 bits of entropy per sample given my the Shannon entropy formula and 0.5 using the min-entropy formula. The only attempt to skew the data was using two sets of recordings one having music played into it and one with ambient background noises. They did find correlations between randomness generated from the music recording, but stated that this could be corrected with the proper post-processing. The authors gave no mention to the power draw of each method. It was also mentioned that manufacturers of both microphones cameras are continuously trying to decrease the noise via both hardware and proprietary (hidden) software. The tests mentioned in [5] were performed in 2007 on dated hardware.

Accelerometers have been studied as a potential source of entropy in low-power and embedded devices. J. Voris et. al. analyzed the use of accelerometers in an Intel WISP RFID tag [9]. They conclude that even when stationary, accelerometers are sensitive enough to provide a good rate of entropy; they even test the device when at rest on industrial dampening material. They show that accelerometers are highly resistant to bias, arguing that the best attack vector to reduce randomness is to leave the accelerometer stationary (i.e. movement only serves to increase the randomness rather than degrade it). While it is conceivable that an attacker may be able to saturate the device's output by applying an extraordinary force (say, in a centrifuge), an accelerometer lacks the potential for bias that plagues other devices on the phone. Although gyroscopes have not been studied as a means for random number generation, they are a similar device in principal to the accelerometer, striving to achieve a high level of sensitivity. In addition, these two devices have become ubiquitous on modern phones and are extremely cheap to include in any system. Because of the low potential for bias and the widespread use of these devices, they are an attractive means for adding entropy to a random number generator.

B. Problem Statement

Due to the increasing number of mobile platforms and their use of cryptographic algorithms and random number generators, there is demand for quick and power efficient random number generators in the mobile environment. Specifically on

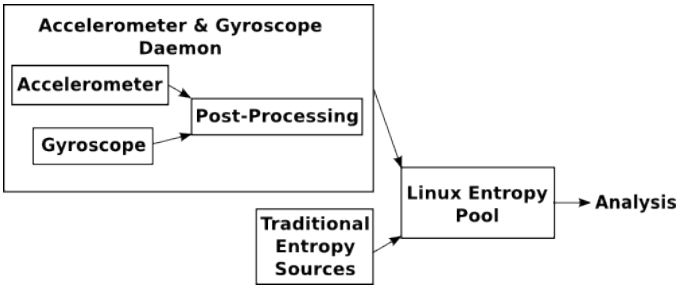

Fig. 1. Block diagram of the proposed solution. Our daemon will poll from the accelerometer and gyroscope (when requested), perform post-processing and feed random bits into the Linux entropy pool via system calls.

our target hardware, Qualcomm's APQ8060 Dragonboard, we have access to the random and urandom devices given via the Linux kernel. The random device is the true random number generation option in the Linux environment, it is therefore a safe baseline for comparison as it is fairly consistent between Android devices. There are issues with this current solution. Firstly, a performance decrease can be witnessed when the entropy pool, used by random, is depleted to 128 bits. This is may be more prevalent in Android's Event Loop software model, as there isn't a clear multi-tasking structure. Another flaw comes to the scene when programmers and users try to circumvent this performance drain, by either filling the entropy pool with a none random bit-stream or by using the urandom device where the random device is more secure. We witness the effects of these issues in sources like [6], which found that 4% of 6.6 million certificates and PGP keys use the same RSA modulus and therefore possibly have the exact same private key as someone else. The authors of [4], exposed poor random number generators in EMV machines. Companies, such as Qualcomm, provide hardware random number generators in their System on a Chip (SoC) architectures to combat these drawbacks. Therefore there is a clear need for a low power solution that is applicable to existing and future mobile devices on the market.

II. Proposed Solution

Although mobile phones have many different devices on-chip that could be potentially used as sources of randomness for a random number generator, most of these devices have the potential for strong bias. Additionally, because of the operating environment polling these devices with high frequency can cause serious system performance and energy issues. Therefore it is necessary to not only use devices that have high quality randomness, but to use those devices in an intelligent manner so as to not overwhelm the computational and energy resources of the device.

We propose the implementation of a daemon process that polls data from the accelerometer and gyroscope (when the entropy pool is running low), processes input data to eliminate sources of bias, and feeds the data into the entropy pool. Figure 1 shows the architecture for our daemon. Android built on Linux exposes standard interfaces to the hardware sensors for data retrieval; additionally, Linux provides system calls to feed random data into the entropy pool. Our daemon will interact with both of these interfaces.

There are many types of post processing that can be implemented to increase the uniformity of the random numbers generated. Hash algorithms like SHA-3 [3] can be used to eliminate bias. Similarly, Barak et. al. propose a technique whereby random numbers are pulled from several independent devices [1]. Two of these numbers are multiplied together and added to a third number to strengthen the randomness. Barak et. al. propose another technique where random numbers are multiplied by a Toeplitz matrix to approach a uniform distribution [2]. These techniques will be implemented and compared for their effectiveness and computational complexity.

Random numbers generated by our daemon will be compared to the stock Android/Linux random number generator (which includes an on-board hardware RNG that may not be present on all mobile devices). These two sources will be compared on three dimensions: quality of random numbers generated, energy draw from using the random number generator, and computational resources used by random number generation. The DIEHARD [7]/NIST [8] benchmarks will be used to evaluate the quality of the random numbers generated; synthetic benchmarks will be written to evaluate the energy draw and compuational compmlexity of the random number generators.

A. Motivation

B. Expectations

III. Plan for Development

A. Risks and Unknowns

The work for the proposed solution will be broken down into three phases. The first phase will involve the actual implementation of the daemon and benchmarking the stock random number generator with the DIEHARD and NIST standards. Two people will work on the daemon while the other two work on benchmarking and post processing. The second phase will involve collecting data and testing the solution. Because the Dragonboard is a development platform and is not intended to be carried on a person, data will need to be collected from accelerometers on production mobile devices. This data will be compared with the Dragonboard to determine the increase in entropy from moving the acceleromter more. During this phase, the implemented solution will be benchmarked with DIEHARD and NIST for comparision to the stock random number generator. Two people will collect the accelerometer data while the third and fourth benchmark and measure power usage of the solution. The final phase of the project will be to prepare the report. Major sections of the report can be worked on independently, in parallel. The group will meet to discuss and finalize the conclusion section and submit the report.

All of the tasks are laid out on the Ghantt chart shown

For any project, it is important to understand that there are risks and unknowns that will need to be understood and mitigated during the course of the project. By assessing what may become a problem during the process of the project, those problems can be consciously avoided or dealt with early on.

One unknown that is central to this project is the entropy of the accelerometer. Based on the findings in [9], it appears that accelerometers are good sources of entropy, it is possible

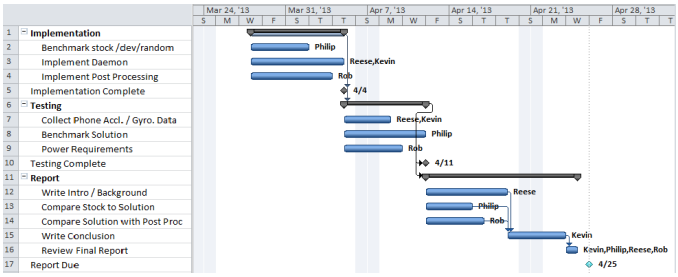

Fig. 2. Assigned tasks and estimated time of completion

that the accelerometer on the Dragonboard violates some assumption or that technology has changed such that modern accelerometers are no longer good sources of entropy. This can be tested using several randomness tests, such as DIEHARD and the NIST entropy tests, which will be essential to do before moving on with any sort of analysis. These tests will show if the sensor data appears to have good entropy or not. In the case that it does not, it would invalidate the remaining work for the project.

A potential problem is the existence of the hardware random number generator that is already on the board. Depending on how good that hardware is, it could potentially dwarf any results that this project generates. Even so, the results should be valuable, both as a look at the prospect of using accelerometers in future designs for hardware random number generators as well as for providing better entropy to existing platforms which do not have hardware random number generators.

If the data from these sensors is already being incorporated into the entropy pool, it would make this work largely a repetition of that previous work. Upon first inspection this does not appear to be the case, though further research is needed to ensure that this is indeed novel work.

IV. Conclusion

The conclusion goes here.

References

[1] Boaz Barak, Russell Impagliazzo, and Avi Wigderson. Extracting randomness using few independent sources. *SIAM Journal on Computing*, 36(4):1095–1118, 2006.

[2] Boaz Barak, Ronen Shaltiel, and Eran Tromer. True random number generators secure in a changing environment. *Cryptographic Hardware and Embedded Systems-CHES 2003*, pages 166–180, 2003.

[3] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Keccak specifications. *Submission to NIST (Round 2)*, 2009.

[4] Mike Bond, Omar Choudary, Steven J Murdoch, Sergei Skorobogatov, and Ross Anderson. Chip and skim: cloning emv cards with the pre-play attack. *arXiv preprint arXiv:1209.2531*, 2012.

[5] Jan Krhovják, V Matyas, and P Svenda. The sources of randomness in mobile devices. *In Proceeding of NORDSEC*, pages 73–84, 2007.

[6] Arjen K Lenstra, James P Hughes, Maxime Augier, Joppe W Bos, Thorsten Kleinjung, and Christophe Wachter. Ron was wrong, whit is right. *IACR eprint archive*, 64, 2012.

[7] George Marsaglia. Diehard: a battery of tests of randomness. *See http://stat. fsu. edu/ geo/diehard. html*, 1996.

[8] A Rukhin, J Soto, J Nechvatal, M Smid, E Barker, S Leigh, M Levenson, M Vangel, D Banks, A Heckert, et al. Nist special publication 800-22. *A statistical test suite for random and pseudorandom number generators for cryptographic applications*, 2001.

[9] Jonathan Voris, Nitesh Saxena, and Tzipora Halevi. Accelerometers and randomness: perfect together. In *Proceedings of the fourth ACM conference on Wireless network security (New York, NY, USA, 2011), WiSec*, volume 11, pages 115–126, 2011.