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. Background

B. Related Work

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 [1], 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 5C 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 [1] were performed in 2007 on dated hardware.

C. 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 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 [**?**], 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 [**?**], exposed poor random number generators in EMV machines. Companies, such as Qualcomm, provide hardware random number generators in

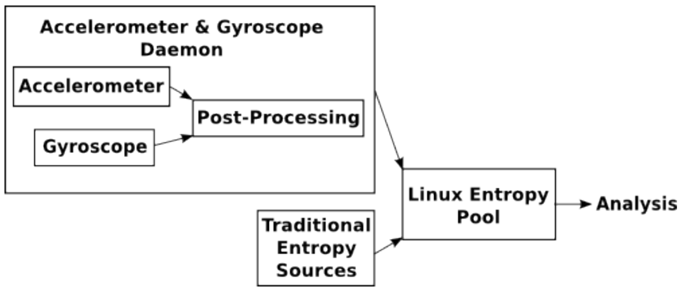

Fig. 1. Block diagram of the proposed solution. Our daemon will poll from the accelerometer and gyroscope (when requested), perform post-processing and feed random bits into the Linux entropy pool via system calls.

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

Mobile phones have become diverse systems-on-a-chip, incorporating many types of devices onto one platform. Devices such as touchscreens, cameras, microphones, battery levels, etc. have been investigated as additional sources of entropy on these systems, as they have become ubiquitous. However, a serious drawback of many of these devices is the potential for strong bias; for example, a microphone can be biased by a loud tone that overwhelms the sensor and causes it to generate the same saturated values for subsequent samplings. Being able to strongly influence these devices is a serious cryptographic concern - as such, it is important to find a device or devices that can produce high-quality random numbers at fast bitrate.

As discussed in [**?**], accelerometers are a good candidate for this role. Because of the highly sensitive nature of accelerometers, they are resistant to a variety of potential sources of bias. J. Voris et. al. provide a detailed discussion of this resistance, showing that most attempts to produce bias actually increase the randomness of the device. They show that even at its lowest entropic level (when the accelerometer is at rest on industrial dampening material), it still produces high-quality randomness. Additionally, gyroscopes are another potential source of entropy as they are designed to achieve the same level of sensitivity. Thus, both of these devices are attractice ways to add high quality randomness into the entropy pool.

We propose implementation and analysis of a daemon that, when awoken, reads from the accelerometer and the gyroscope, performs post-processing to increase the quality of the data and adds the data to the entropy pool. We will analyze the daemon in regards to power consumption, quality of the random numbers generated, and speedup of random number generation. Figure 1 shows the overall architecture for our daemon. It will perform device polling and post-processing to produce data, then will use system calls to add entropy into the Linux entropy pool. Finally, various synthetic and real-world benchmarks will be used to analyze the speed of random number generation, quality of generated numbers, and power draw from using the daemon.

A. Figure

B. Motivation

C. Expectations

III. Plan for Development

IV. Conclusion

The conclusion goes here.

Acknowledgment

The authors would like to thank...

References

[1] Jan Krhovjak and Petr Svenda. The sources of randomness in mobile devices. 2007.