True Random Generation on Mobile Phones

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

At the core of the majority cryptography operations, there exists a need for true random numbers. True random number generators (TRNG) can be costly to produce. The fastest known TRNG was produced in 2010 at Bar-Ilan University [**?**], claimed to generate at a rate of $300 Gbits^{-1}$. This method makes use of a high derivative of the digitized chaotic laser intensity. However this paper targets a low power devices with limited access to such highly noisy devices. We aim to present a practical approach of a TRNG in a power friendly method. There are many existing TRNG implementations both in software and hardware. The target platform of this paper, and all Android based devices, depends on the randomness provided by the Linux kernel.

A. Related Work and Background

The Linux kernel, used by the Android system, has an entropy pool to provide randomness. This pool is populated via environmental noise from device drivers, timing differences between interrupts, and other various sources. These sources are treated as true random number generation and therefore create entropy suitable for key generation and other cryptography. Access to this pool is granted through the random device located at /dev/random [**?**] and through the urandom device, located at /dev/urandom. These two interfaces read straight from the pool until a limit of 128 bits of remaining entropy is reached. If the pool is drained to below this bit limit, and if the application reading from the pool is using the random interface, then it will block until more entropy is added to the pool. The urandom interface provides "unlimited" entropy by rehashing existing entropy in the pool. The urandom device does not block, but forgoes true randomness and uses and feeds psuedo random data to the application. It is stated that this method is theoretically vulnerable to cryptographic attacks.

In the Google Play Store there exists an application entitled Seeder. The developers of seeder are trying to speed up the Android experience by removing the need to ever block on the random device. Their solution to provide entropy even when the random device's 128 bit limit is reached by to piping 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 (as of the date this paper was written) , 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 [**?**], the two main sources analyzed were the camera and microphone. The authors of this paper were unable to generate definite rates 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 when the camera guard was closed. However there were also some vulnerabilities, an attacker could use a halogen light bulb to saturate the Charge-Coupled Device (CCD) to produce only 255 values for R, G, and B values of each pixel. This would highly bias the randomness produced by the camera. 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 [**?**] were performed in 2007 on dated hardware.

Additional research has also explored the use of a mobile device's touch screen as a source for random number generation. One particular paper [**?**], discussed the use of an input application presented to the end-user where they would move their finger to generate coordinates. These coordinates were filtered to only use the lower eight bits of each value and produced good entropy. However, this solution is only good for generating random data infrequently. An example use might be generating a password or key that doesn't change often. For encrypting session data, such as Transport Layer Security

(TLS), it would require more interaction from the end-user and be inconvenient.

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 [?]. 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 we targeted Android based smart phones,this grants us access to the random and urandom devices given via the Linux kernel. The random device is the true random number generating option in the Linux environment, it is therefore a safe baseline for comparison as it is fairly consistent between Android devices. There are several issues with this current solution. Firstly, a performance decrease can be witnessed when the entropy pool, used by the random device, is depleted to 128 bits. This 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 when 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, used in chip and pin payments. 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 true random generator that is applicable to legacy 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.

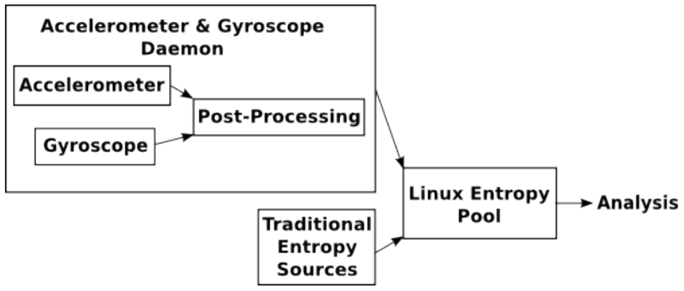

Fig. 1. Block diagram of the proposed solution. Our daemon will poll from the accelerometer and gyroscope (when requested), perform post-processing and feed random bits into the Linux entropy pool via system calls.

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. Fig. 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 [?] can be used to eliminate bias. Similarly, Barak et al. propose a technique whereby random numbers are pulled from several independent devices [?]. 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 [?]. 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 [?] / NIST [?] benchmarks will be used to evaluate the quality of the random numbers generated; synthetic benchmarks will be written to evaluate the energy draw and computational complexity of the random number generators.

Efficient generation of random numbers, with high entropy, is essential for mobile devices, however power must always be used carefully as to not drain the battery too quickly. Any peripheral device, such as an accelerometer, will use more power when it is collecting data. The APQ8060 Dragonboard uses a Kionix KXSD9-1026 accelerometer which draws typically

0.3 μAmps at 2.6 Volts while in standby [?]. However, full operating power draws more than 200 μAmps. Running the accelerometer for any length of time, at a high sampling rate, may draw more power than other sources of entropy. As a result, the implemented daemon will be able to sense when the entropy pool is full and pause its collection until it senses that the pool has been depleted. This should prevent the battery from being drained excessively as a result of this method of entropy gathering.

A. *Expectations*

From the findings of [?], accelerometers appear to be highly random. A high rate of sampling shows that they are fairly fast and should be able to provide a high rate of random bits per second when they are needed. The specification sheet for the accelerometer found on the Dragonboard shows that it is also efficient for power consumption.

The apparent randomness of the accelerometer and gyroscope, accompanied with the relatively low power nature of the components as well as their ubiquity on smart phones, tends to suggest that they would be an excellent source of entropy for mobile platforms. A solution that pulls from these sensors should be easily deployed to several different platforms, including those that do not have the benefit of an on board hardware random number generator. As quality random numbers are necessary for most of the common cryptographic operations, such a system will be able to bring faster secure communication to more devices.

III. Plan for Development

In order to implement and analyze the proposed solution, work has been broken down into three phases: the implementation phase, the testing phase, and the report phase. The implementation phase focuses on doing initial benchmarking and developing of the proposed daemons. The testing phase will consist of analyzing the data that is generated from the implementation phase. The report phase is where the final report and presentation are created and all of the results summarized.

The work timeline is layed out on the Ghantt chart in Fig. **??**. This chart shows who will be performing which tasks, and when they will begin and finish those tasks. It also shows how the stages are dependent upon the previous stage being completed. Breaking down the tasks like this helps the project move forward at a reasonable and planned pace.

In the implementation phase of the project, one member of the group will benchmark existing randomness sources on the board in order to generate a baseline against which any developed randomness solution can be measured. Two members of the project will develop the daemon discussed earlier to take presumably random data from the accelerometer and introduce it into the entropy pool of the Linux kernel. One member of the group will develop several post processing implementations which will hook into the daemon to provide better entropy to the kernel than simply taking all of the raw data. Most of the development will be done in this phase of the project, as the rest of the project will be focused on collecting and analyzing data and building the conclusions for the final report.

The testing phase of the project is where data will be collected using the solution developed in the implementation phase. Two members of the group will collect accelerometer and gyroscope data for later analysis, to show that this data has consistent properties. As a development platform, the Dragonboard is not intended to be carried on a person, as a traditional cellphone would be. This means that the team members collecting data will not only collect data on the board, to show the effects of relative stillness on the sensors, but also collecting data on personal cellphones. This will require some analysis to show that these tests properly approximate what they would have been using a mobile Dragonboard. The member of the project who benchmarked the stock randomness solution will now follow a similar procedure to benchmark the developed solution. This will allow for direct comparison between the stock and developed solutions, showing what benefit the developed solution provides. A member of the project will benchmark the developed system in terms of its power draw, running the system while sampling the current draw of the entire system, to show the effects that this implementation has on the battery life of the device.

During the review phase of the project all of the data will be analyzed so that the final report and presentation can be prepared. The entire paper will be split between all of the members of the project. It is partitoned in such a way that the members are writing the sections that they would have had most exposure to during the initial two phases of the project. After indvidual contributions have been made, the group will meet as a whole to discuss the paper, including finalizing the conclusion. This will culminate with submitting the deliverables.

A. *Risks and Unknowns*

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 [?], it appears that accelerometers are good sources of entropy, it is possible 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 the DIEHARD and 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.

During the testing phase of the project, members of the project team will perform an experiment to show what sort of data is experienced over the course of a couple of days. As it is not reasonable to carry the Dragonboard over the course of a daily routine, some of the data will need to be collected on personal cell phones. It could potentially be a problem if no available cell phone has a similar accelerometer or gyroscope as this would introduce an additional variable into the analysis. Research and experimentation will have to be done to discover if, and show, the sensors in the selected cell phones will be a reasonable stand in for the Dragonboard. If nothing matches closely enough, a different method of collecting longer term real world data will have to be devised.

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. Deliverables

At the culmination of this project, the deliverables will include a detailed report, the daemon source, post processing source, and any scripts used in development. The report will include an analysis comparing the results with the stock solution in terms of entropy, speed, and power. The results will be presented, summarizing the findings of the report.

This work will show that the potential exists to use common peripherals on mobile devices to supply random bits to fill the entropy pool to feed the RNG. This will be beneficial to developers who need to ensure a consistent stream of cryptographically high-quality random numbers. The end-user may not typically be aware of the presence of the RNG on their system, they do notice slow downs as a result of encryption application sessions waiting for adequate entropy to be secure. The proposed solution will allow application developers to provide those applications with constant streams of high entropy bits preventing these stalls from ever occurring, even on low end hardware that would not have a hardware random number generator included.

V. Experimental Setup

VI. Results

VII. Future Work

In this paper we present analysis for two sensors on an Android smartphone, the accelerometer and gyroscope. However, these devices are becoming increasingly diverse in terms of capabilities; many newer devices include near-field communications, GPS, magnetometers, etc. Additionally, phones contain cameras, microphones and touchscreens, all of which have been studied as sources of random bits. These sensors and devices could potentially be leveraged to increase the diversity of sources and the possibly increase the entropy in the produced data. Investigating the power efficiency of the system related to the number of devices utilized and the sampling rate of these devices would also be another area to explore, as mobile devices are at the mercy of the life of the battery.

One feature of the Android ecosystem, which is simultaneously an advantage and a disadvantage, is the variability in phone designs. While we have evaluated one particular set of sensors on a particular model of phone, there are innumerable designs for which the data obtained from the sensors varies widely. Initial testing of several newer models of phones show raw data from the accelerometer and gyroscope may provide higher-quality random bits than the values obtained from the Nexus S. Evaluating a wide range of these devices could provide better insight into whether using these devices for true random number generation is a viable means for augmenting the sources of entropy for the Linux entropy pool.

Finaly, further investigating different types of post processing may also provide a means for obtaining higher quality bits from the devices. While we have presented several basic forms of post processing (bit filtering and von-Neumann whitening), many authors have published work in the field of randomness extraction. Barak et. al. propose a mathematical model, based on the influence an attacker may have over the source of random data, and an approach to constructing a randomness extraction function which is guaranteed to provide good random bits [**?**]. Additionally, Barak et. al. propose a method for combining data from independent sources to provide higher quality random bits [**?**]. Both of these approaches could be leveraged to increase the quality of the data obtained from the sensors.

VIII. Conclusion