SOMAIYA VIDYAVIHAR UNIVERSITY

Internal assessment Report on

End-to-End System Design for DRAM-based True Random Number Generators

By

16010121022: **Jagjit Singh Bhumra**
16010121043: **Vishrut Deshmukh**
16010121045: **Pargat Singh Dhanjal**
16010121168: **Vatsal Sanchala**

For the subject

Applied Cryptography (Honors in CSF)

Department of Computer Engineering
K. J. Somaiya College of Engineering
(Constituent College of Somaiya Vidyavihar University)
Academic Year 2022-23

Topic chosen: **End-to-End System Design for DRAM-based True Random Number Generators**

Paper selected:
https://people.inf.ethz.ch/omutlu/pub/DR_STRANGE_EndtoEnd-DRAM-TRNG_hpca22.pdf

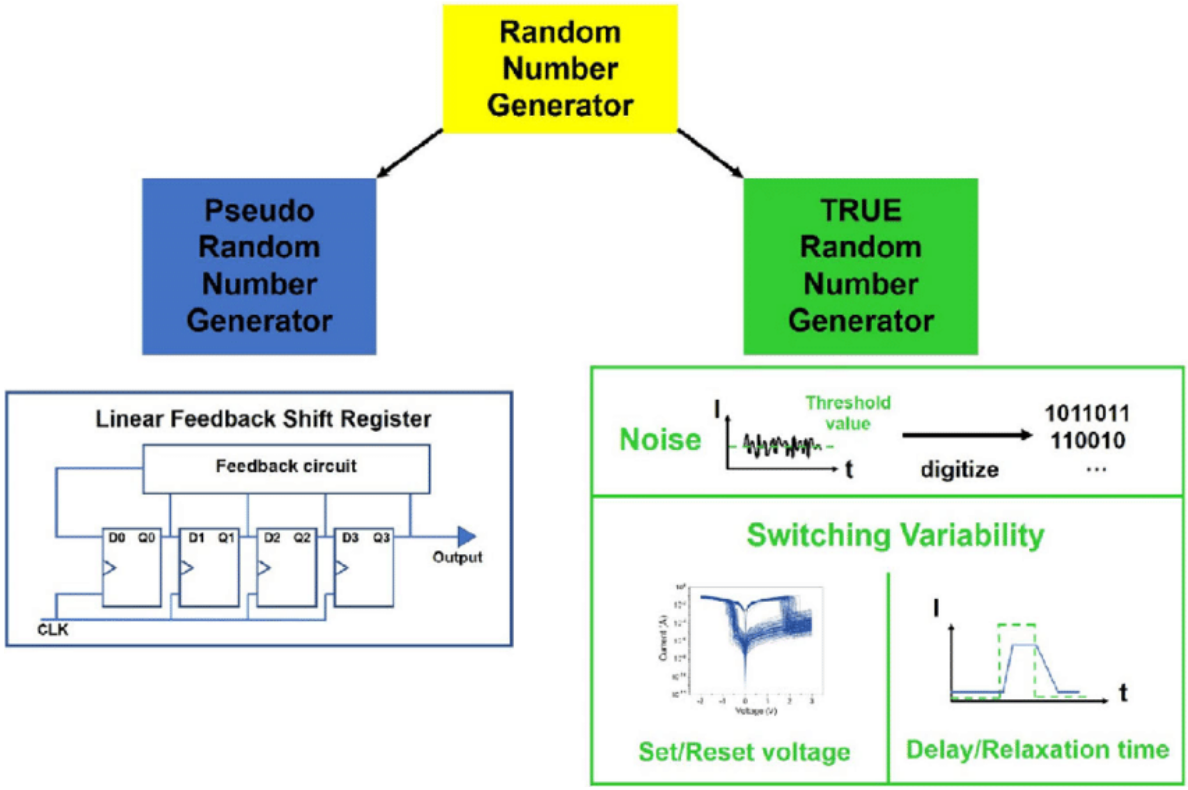
Year of publication: 2022

Introduction

Random number generation is a widely used field of computer domain. Whether we take the example of passwords, hashing seeds, key generation seeds or just plain encryption, its use extends far and wide. Nowadays, not only military-grade and corporate security - even the common man is dependent on companies using random numbers for encryption, to keep all their data safe and secure.
All the responsibility of keeping these random numbers “actually random” is upon the concept of random number generators, or RNGs. Random number generators are basically programs that can take input in the form of a seed - physical or virtual - and then generate a series of numbers using built-in algorithms. The randomness of a generated number depends upon this seed, and using weak or repetitive seed numbers could weaken the whole random generation algorithm. Majority of what the world uses for random number generation is PRNGs, or pseudorandom number generators. These depend on a virtual seed (stream of numbers) fed into the system to generate random numbers. The other recent entry to random number generation are TRNGs, or truly random number generators. These depend on a physical entropy change, and are “truly” random as their name suggests.For PRNGs, in theory, if an attacker gets access to the seed used to generate the random number, he could essentially use it to generate the same number. This poses a valid problem for the modern world. To understand more about this, we need to have a look at the main differences between TRNGs and PRNGs.

TRNGs vs PRNGs

The two types of random number generators are: PRNG (Pseudo Random Number Generator) and TRNG (True Random Number Generator). PRNGs intentionally generate a random number, not a complete random number, and the number is usually generated by a software. TRNGs, on the other hand, generate a true random number and is generated primarily by hardware, or using a physical entity. For this reason, the random numbers generated by TRNGs are hard to predict because TRNG is generated based on a physical source - that is, it is difficult to predict a random value. Therefore, the random number generated from TRNG is a safe method because it is difficult to generate the same value. However, PRNG is a deterministic system so the generated random number cannot be guaranteed to be safe, since in theory, losing the seed can mean losing the randomness of a number. Hence, there is a problem that the attacker conjectures the same key derived from the same random number by generating using the same initial seed. In addition, it is impossible to generate a completely random number using only mathematical algorithms.



Proposed Solution

A perfect system that can generate truly random numbers was proposed in the research paper. It is named DR-STRaNGe, which stands for End-to-End System Design for DRAM-based True Random Number Generators. What problems it aims to solve, and how it aims to solve them, is detailed in the report below.

Problem 1

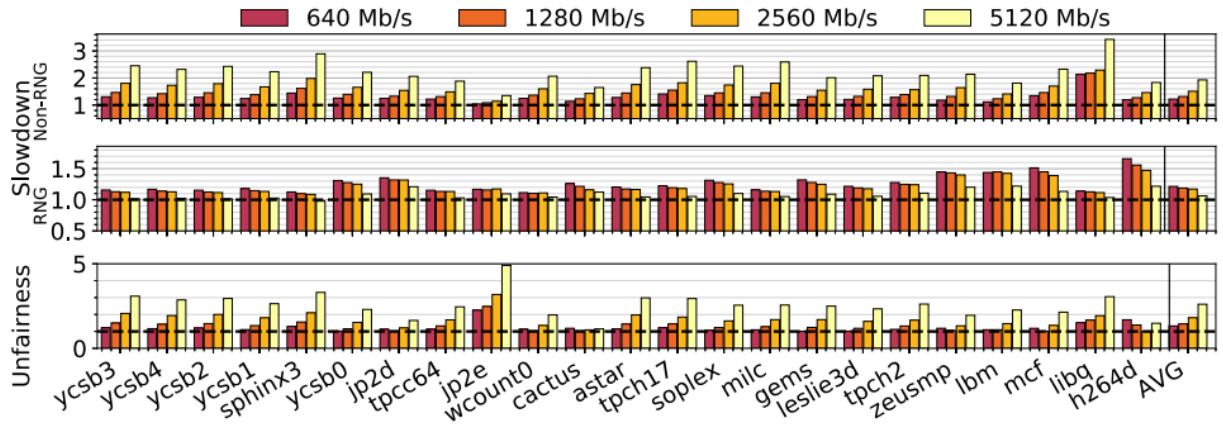
- The first problem encountered with DRAM-based True Random Number Generators is that while it is great for generating “actual” random numbers, it often slows down the whole computer system.
- This is because Random Number Generation (RNG) applications, and non-RNG applications running side-by-side puts heavy strain on the system DRAM, sometimes so much that it causes all operations to freeze.
- This is a severe disadvantage in any computer system, and one that once resolved, could show high advantages.

Problem 2

- The second problem that plagues these types of TRNG systems, is the biased prioritization for RNG vs non-RNG applications, that is done by the computer system.
- This makes any Random Number Generation computation unfair compared to other non-RNG workloads, so that other applications’ performance degrades to further and further extents.
- The computer system becomes unusable for anything other than random number generation, proving to be a major flaw which means this design cannot be used commercially (need a separate computer system for RNG applications then).

Problem 3

- The final common flaw found in systems like these is the high latency of DRAMs of computer systems in general.
- Not everyone has access to computers built with the highest specifications, and this proves to be a real problem in random number generation.
- High-latency-ridden DRAM systems can cause severe delay in random number generation, this in turn leads to greater delays down the process.



The figure above shows the slowdown experienced, as well as unfairness index, of RNG vs non-RNG applications over a wide variety of benchmarking formats, as well as a range of different throughput speeds.
(Higher throughputs causes greater unfairness and more slowdown, across the board)

Solution 1

The need to design a system that solves the problem of RNG vs non-RNG application interference was needed, to prevent system slowdown. This was achieved using a multitude of methods, to build a Random Number Buffering Mechanism:

1. **Simple Buffering Mechanism**

To reduce the application interference, a random number buffer is introduced into the memory controller. The idea behind this is to only use those cycles where the DRAM is idle, for any RNG computation, and store the random bits generated to this buffer in advance. Doing this minimizes any interference between the normal applications running on the computer system and RNG operations, and also reduces latency problems due to prestoring of generated random bits.

2. **DRAM Idleness Predictor**

This particular mechanism waits and predicts a suitable period of idle time, enough to generate 8 random bits and store it in the buffer. This prediction is important since using very short idle times still end up interfering in normal system operations. The predictor helps choose longer idle time periods, and neglect short ones, so that RNG and non-RNG applications never run concurrently. However, these predictions also use up quite a bit of computational power, which is why the solution includes one more major feature.

3. **Reinforcement Learning**

Since we need to minimize the unnecessary use of DRAM, constantly running the idleness predictor could lead to greater slowdown than even initially. To curb this, a reinforcement learning feature is implemented in DR-STRaNGe - this means the system “learns” for what and how much DRAM is used, during different parts of the day. SInce each user has a different work area, and a variety of processes running, the system can personalize the random number generation according to each user’s needs and conditions.

Using all the above methods in harmony results in great improvements in overall TRNG and system performance, as shown in the results section. This is how application interference was solved, by differentiating between RNG and non-RNG memory calls.

Solution 2

A system was needed such that it reduces or prevents the unfair prioritization of RNG vs non-RNG requests. This was needed because the memory buffer implemented may not always have the random bits ready, so an RNG-aware memory requests scheduler is constructed to take care of this. This scheduler is used in cases when the system may need to stall specific requests, to allow other operations to take place. The RNG-aware scheduler was implemented keeping the following status modes in mind:

1. **RNG-Prioritization**

In this mode, RNG requests were given higher priority compared to normal operations. This mode is used in cases where there is a dire need for more random bits to be generated, or the memory buffer to be filled - in such cases there is no other option available but to stall non-RNG requests and give way to the RNG ones.

2. **Non-RNG-Prioritization**

In this mode, normal requests are prioritized over RNG requests. This is done in cases when RNG-prioritization has resulted in long stall times for normal operating of the system - resulting in the user applications freezing. To prevent this, RNG requests are stopped, and normal operations are given a bit of overhead to resume their work.

3. **Equal-Prioritization**

In this mode, both RNG and non-RNG requests are given equal priority. This mode is used generally all the time, since it prevents either of the request types to fall back and breakdown altogether. Giving equal priority helps in most cases, and whenever the need, the RNG-aware scheduler can switch back to one of the other 2 modes to normalize the system operations in real-time.

This RNG-aware scheduler, combined with all these modes, provides enough overhead to prevent any operations slowdown, even if there are several memory-intensive applications running.

Solution 3

The problem of latencies is solved to quite an extent using the first 2 solutions, since the memory buffer stores random bits in advance, with the help of DRAM idleness predictor - combined with the prioritization system, giving it a boost.

However, there also needs to be an interface to communicate between the software and our TRNG system. DR-STRaNGe implements this in a great way, by replacing the usual getrandom() call (to generate random bits), with a custom call that communicates directly with the DRAM, thus reducing the latency even more than before. The interface, by providing direct communication, also increases the security of the Random Number Generator, since the random bits are invoked and sent directly to the system, as and when required.

Advantages & Results:

- DR-STRaNGe improves performance of non-RNG applications and energy by 17.9% and 21%, respectively, compared to the commonly-used baseline memory request scheduler design
- It improves system fairness by 32.1% on average when generating random numbers at a 5 Gb/s throughput.
- It reduces energy consumption by 21% compared to the RNG-oblivious baseline design by reducing the time spent for RNG and non-RNG memory accesses by 15.8%.
- DR-STRaNGe is the first end-to-end system proposed that aims to correct the 3 most common flaws found in modern DRAM-based TRNG systems.
- The system is tested with 2 famous TRNG mechanisms, and the results show that they work better when run with DR-STRaNGe, than without.
- Each new feature implemented DR-STRaNGe was shown to actually improve performance drastically, and adding them all up meant that TRNG systems underwent at least a twofold improvement.
- Not only performance-enhancements, this end-to-end system managed to pack all of this within a marginal area overhead of 0.0022mm^2 only.

Conclusion:

By researching about DR-STRaNGe, and DRAM-based TRNGs in general, we learnt a lot about how random numbers are generated in the real world. Pseudorandom number generators are everywhere nowadays, however they're a lot easier to predict and steal than we used to think. The modern world requires True Random Number Generators, ones based on change in physical entropy of some kind, to keep our data and secrets secure.
DR-STRaNGe provides this power of true randomness to the common man, enabling us to use this technique in everyday computer systems. This is something that was thought to be fictional just mere years ago. The fact that DRAM-based TRNG systems have become a reality, and may very soon become the norm as well, is nothing short of a boon for the technology world.

We would like to thank Prof. Swati Mali for guiding us throughout this semester, helping and supporting us to learn Applied Cryptography in a fun and meaningful way. By giving us short insights into how the real world around us always has cryptographic elements embedded into it, she made it our pleasure to deepdive into this subject.