Trail of Bits Blog

State of the Art Proof-of-Work: RandomX

- **POST**
- JULY 2, 2019
- LEAVE A COMMENT

RandomX (https://github.com/tevador/RandomX) is a new ASIC and GPU-resistant proof-of-work (https://en.wikipedia.org/wiki/Proof_of_work) (PoW) algorithm originally developed for Monero (https://www.getmonero.org/), but potentially useful in any blockchain using PoW that wants to bias towards general purpose CPUs. Trail of Bits was contracted by Arweave (https://www.arweave.org/) to review this novel algorithm in a two person-week engagement (https://github.com/trailofbits/publications/blob/master/reviews/arweave-randomx.pdf) and provide guidance on alternate parameter selection. But what makes it unusual and why should you care about it?

Standard Proof-of-work

In classical PoW algorithms (such as hashcash (https://en.wikipedia.org/wiki/Hashcash), used in Bitcoin), the core is typically a cryptographic hash function where the only variable is the data input to the function. To achieve a target "hardness" a number of zero bits are required as the prefix of the hash output. Each zero bit added doubles the difficulty of mining. However, this type of algorithm is highly amenable to acceleration via ASICs and GPUs because a fixed set of operations are performed on all input with limited memory requirements. This is undesirable.

Why do we care about ASIC resistance?

Blockchain mining is ideally a heavily decentralized task with no singular entity controlling a significant amount of the hashing power. Blockchains are susceptible to 51% attacks, where a malicious majority can override the global state, e.g., allowing double-spends. If an ASIC can be built that significantly improves mining efficiency over a general purpose CPU, then

economic factors will disincentivize CPU-based mining. The result of this can be seen in the Bitcoin network, where ASIC manufacturers have built large-scale mining farms and a handful of entities control a shockingly high percentage of the hash rate.

For the last several years, ASIC manufacturers have shown great capacity for rapid design and fabrication of ASICs. A project that wants to be ASIC-resistant (without switching to a proof-of-stake (https://en.wikipedia.org/wiki/Proof_of_stake) model, which has its own set of tradeoffs) must therefore seek to take advantage of the specific strengths a general-purpose CPU possesses over a hypothetical ASIC.

This desire has led to a significant amount of research (https://eprint.iacr.org/2017/225.pdf) around ASIC resistance. RandomX represents a concrete implementation of the most modern ASIC-resistant ideas as applied to cryptocurrency.

How RandomX works

The core of RandomX is the concept of randomized execution. Put simply, we want to execute a series of random instructions to take advantage of a general-purpose CPU's flexibility with dynamic code execution. The RandomX developers have extensively documented their design rationale (https://github.com/tevador/RandomX/blob/master/doc/design.md) and provided a specification (https://github.com/tevador/RandomX/blob/master/doc/specs.md) with a more rigorous explanation, but with some simplification the algorithm does the following:

Step 1

A data structure called the Cache is created using argon2d (https://github.com/tevador/RandomX/blob/master/doc/design.md) with the input key K. Argon2d was originally designed as a memory-hard password hashing function. Computers generally have large pools of fast memory available to them, but memory is expensive on an ASIC. Requiring large amounts of memory is one of the most common defenses against specialized hardware. Argon2 uses a variety of techniques to ensure that a large (configurable) quantity of memory is used and that any time/memory tradeoff attacks are ineffective. You can read more about them in the argon2 specification (https://github.com/P-H-C/phc-winner-argon2/blob/master/argon2-specs.pdf).

Step 2

The Dataset (a read-only memory structure) is expanded from the Cache. Datasets are designed to be a large segment of memory that contain data the virtual machine will read. There are two values that control the size of the Dataset (RANDOMX_DATASET_BASE_SIZE and RANDOMX_DATASET_EXTRA_SIZE). Together, they place an upper bound on the total memory the algorithm requires. Extra size is used to push the memory slightly beyond a

power of two boundary, which makes life harder for ASIC manufacturers. The actual Dataset generation is performed by loading data from the Cache, generating a set of SuperscalarHash instances, and then invoking those instances to get a final output. SuperscalarHash is designed to consume power while waiting for data from DRAM. This hurts an ASIC that attempts to compute Datasets dynamically from the Cache.

Step 3

The Scratchpad (read/write memory) is initialized by performing a blake2 hash on the input data and using the resulting output to seed the AesGenerator. This generator uses AES-NI instructions to fill the Scratchpad. Generation of the initial Scratchpad uses AES transformations. This algorithm is already hardware-accelerated on modern CPUs, so an ASIC will gain no advantage implementing it. The Scratchpad itself is a (relatively) large read/write data structure designed specifically to fit in caches that are available in CPUs.

Step 4

Now we get to the core of the algorithm: the randomized programs running on a virtual machine. The VM is executed by building a program using random bytes created using another generator. The RandomX virtual machine architecture (https://github.com/tevador/RandomX/blob/master/doc/design.md#2-virtual-machine-architecture) is carefully designed to allow any sequence of 8-byte words to be a valid instruction. These instructions are designed to:

- Require double precision floating point operations
- Use 128-bit vector math
- Use all four IEEE 754 floating point rounding modes
- Read and write to the Scratchpad, which as previously stated is designed to fit entirely in CPU caches and thus be very fast
- Take advantage of branch prediction via a low probability branch instruction
- Execute instructions using the superscalar and out-of-order execution capabilities of a CPU

Each of these properties is a particular strength of general-purpose CPUs and requires additional die area to implement on an ASIC, reducing its advantage.

The resulting state of the VM after program execution is hashed and used to generate a new program. The number of loops executed in this fashion is configurable but is set to eight by default. This looping behavior was chosen to avoid situations where an ASIC miner might only implement a subset of possible operations and only run “easy” programs on the virtual machine. Since the subsequent program cannot be determined until the current one has been fully executed, a miner cannot predict whether the entirety of the chain will be “easy,” so it becomes impractical to implement a partial set of instructions.

Step 5

Finally, the Scratchpad and Register File (the virtual machine's registers) are hashed using AesHash followed by a final blake2 hash. This step offers no significant ASIC resistance beyond the use of AES instructions, but is included to show the final hashing to a 64-byte value.

What we found

In the course of our two person-week review (https://github.com/trailofbits/publications/blob/master/reviews/arweave-randomx.pdf) we found three issues (two low severity, one informational).

Single AES rounds used in AesGenerator

The AES encryptions described by the RandomX specification refer to a single round of AES, which is insufficient for full mixing of the input data. RandomX doesn't depend on the encryption of AES for its security. Instead, AES is used as a CPU-biased fast transformation that provides diffusion across the output. However, diffusion of bits through the output is dependent upon the number of rounds of AES.

The severity of this finding is "low" because the requisite lack of diffusion requires finding additional bias in almost every step of the algorithm, and then crafting an input that can propagate that bias through the entire chain.

Subsequent to the disclosure of this finding, the RandomX team developed a new AesGenerator4R function that performs four rounds. This functionality has been merged into RandomX as of pull request 46 (https://github.com/tevador/RandomX/pull/46). Four rounds as part of program generation resolves the concerns documented in this issue.

Insufficient Testing and Validation of VM Correctness

The RandomX codebase lacks test coverage validating the semantics of the virtual machine (VM). Trail of Bits devoted half of this engagement (one person-week) to assessing the general security properties of the algorithm implementation. While this effort revealed several code-quality findings, it was insufficient to validate the absence of semantic errors in the VM implementation.

The severity of this finding is "low" because the correctness of RandomX is irrelevant as long as: (1) its output is deterministic, (2) its output is cryptographically random, and (3) its reference implementation is the sole one used for mining in a blockchain. However, any discrepancy between the specification and the reference implementation can lead to consensus issues and forks in the blockchain.

Consider the case where a third-party cleanroom implementation of the RandomX specification becomes popular on a blockchain using RandomX for proof-of-work. The blockchain will fork—potentially at some distant point in the past—if there is even a subtle semantic difference between the miners' implementations.

Configurable parameters are brittle

RandomX contains 47 configurable parameters, including flags for parallelization, memory consumption, and iterations of the initial KDF, memory size of the dataset, sizing of three levels of cache for the virtual CPU, the size and iteration count of programs executed on the VM, and cache access/latency. The default parameters have been chosen to maximize CPU advantage for the algorithm. However, the threat of 51% attacks forces alternate blockchains interested in using RandomX to make different choices for these parameters. These choices must be made without clear insight into which ones may compromise the algorithm's advantages. This brittleness could impede third-party adoption.

Subsequent to this finding, the RandomX team has removed a few unnecessary parameters, written additional guidance (https://github.com/tevador/RandomX/blob/master/doc/configuration.md) about what configuration values are safe to change, and added a new set of checks that prohibit a set of unsafe configurations (https://github.com/tevador/RandomX/blob/master/doc/configuration.md#unsafe-configurations).

Assessment depth

There is a belief in the blockchain industry that many small reviews is a better use of review capital than one large one. This belief is predicated on the notion that every review team will approach the codebase differently and apply different expertise. The supposed diversity of approaches and expertise will provide better coverage and shake out more bugs than having a single team do a deep dive on a given project.

We believe that larger, singular assessments provide better overall value to the client. As a customer, you are paying the assessment team to provide their expert opinion, but like any new employee they need time to ramp up on your codebase. Once that initial learning period is complete, the quality and depth of the assessment rapidly increases. Many large-scale, long-term code assessments do not reveal their most critical or meaningful findings until late in the engagement.

As a client you should hire a single firm for a larger engagement rather than multiple firms for smaller ones for precisely the same reasons you place value on employee retention. Replacing a firm that has domain knowledge will cost time and money if you choose to employ a new firm.

This principle of software assurance is captured well in an old talk from Dave Aitel: The Hacker Strategy (https://www.immunityinc.com/downloads/DaveAitel_TheHackerStrategy2.pdf). Namely, the quality of vulnerabilities a researcher can find is strongly correlated to the time spent. One hour nets you extremely shallow problems, one week significantly more depth, and with one month you can find vulnerabilities that no one else is likely to discover.

This is further discussed in Zero Days, Thousands of Nights (https://www.rand.org/pubs/research_reports/RR1751.html) — the authoritative reference on vulnerability research based on dozens of interviews with experts in the field:

> *The method of finding vulnerabilities can have an impact on which vulnerabilities are actually found. As one example, recent research claims that fuzzers find only the tip of the iceberg in terms of vulnerabilities (Kirda et al., no date). Vulnerabilities can be found via fuzzing in newer or less mature products, or those with simple code bases. For products that have a longer life, are more complex, are popular with a large market share, or are high revenue generators, more people have evaluated the code bases, and finding vulnerabilities often requires more in-depth auditing, logic review, and source code analysis, in order to go several layers deep.*

For example, manually validating the correctness of the RandomX VM (which is the most serious issue we discovered) would take several person-weeks alone. It's highly likely that, at the end of all four audits, there will be no guarantee that the VM implementation is free of semantic errors.

Similarly, analyzing the cryptographic strength of each function in RandomX was achievable within the engagement, but exploring whether there exist methods to propagate potential bias across steps requires a deeper look. Small engagements prohibit this type of work, favoring shallower results.

Current project status

Our two person-week audit (https://github.com/trailofbits/publications/blob/master/reviews/arweave-randomx.pdf) was the first of multiple reviews the RandomX team has scheduled. Over the next few weeks the project is undergoing three additional small audits, the results of which should be published later this month. Once these audits are published and any additional findings are resolved by the RandomX authors, it is the intent of both Arweave and Monero to adopt this algorithm in their respective products in a scheduled protocol upgrade.

By Paul Kehrer Posted in Blockchain (https://blog.trailofbits.com/category/blockchain/), Cryptography (https://blog.trailofbits.com/category/cryptography/)