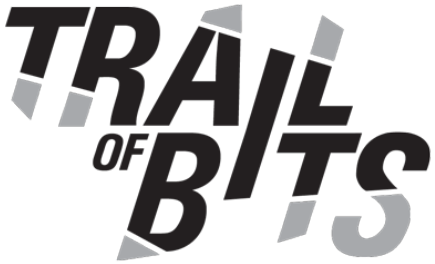

RandomX

Security Assessment

May 28th, 2019

Prepared For:
Sam Williams | *Arweave*
sam@arweave.org

Howard Chu | *Monero*
hyc@symas.com

Prepared By:
Paul Kehrer | *Trail of Bits*
paul.kehrer@trailofbits.com

Will Song | *Trail of Bits*
will.song@trailofbits.com

Evan Sultanik | *Trail of Bits*
evan.sultanik@trailofbits.com

Executive Summary

From May 20th to June 3rd 2019 Arweave engaged Trail of Bits to assess the RandomX proof-of-work (PoW) algorithm implemented in C++. The codebase was assessed as of git commit `378d5de`, corresponding to release 1.0.2. This assessment was conducted using two engineers over two person-weeks.

During the week of May 20th, 2019, Trail of Bits began its assessment of the RandomX algorithm, using a single engineer and focusing on the identification of major cryptographic flaws. During the following week of May 27th, Trail of Bits expanded its assessment to include non-cryptographic attacks, also using a single engineer.

The RandomX algorithm has several complex and novel constructions that make a single week of review insufficient for a complete cryptographic analysis of the system. Trail of Bits focused its efforts on confirming that the use of AES in the algorithm was safe, determining if initial entropy input to the system was preserved throughout the transformations, and determining safe alternate configuration values for Arweave's use.

We found one low-severity issue with the use of single AES rounds in RandomX not providing full diffusion of input bits across the output. We also documented concerns around bias in the output of SuperscalarHash when used in conjunction with non-default parameters in Appendix D. A survey of configuration parameters and which ones are safe to change is included in Appendix E. Appendix F recommends changes for Arweave.

The RandomX codebase lacks sufficient unit and regression tests to validate virtual machine (VM) correctness. The security implications of this omission depends on the way RandomX is adopted, as well as the way RandomX is maintained in the future. If this implementation of RandomX is used for mining a PoW blockchain, then an error in the VM implementation (*e.g.,* a deviation from the specification) will be irrelevant as long as the output of the VM is deterministic. However, if this implementation is ever updated in such a way that the VM semantics change, *or* if a cleanroom implementation of RandomX is ever used to mine blocks on the same blockchain, then this can lead to consensus errors and forking. While no deviations between the VM implementation and the specification were discovered, the scope of this engagement was insufficient to validate that no latent errors exist.

In addition to the security findings, we discuss code-quality issues not related to any particular vulnerability in Appendix B. One such issue was related to undefined behavior in a test edge case. An analysis of the randomness of RandomX is provided in Appendix C.

RandomX is an intriguing, novel hash function specifically designed for PoW blockchains. It should not be used as a general purpose keyed cryptographic hash. Trail of Bits recommends a more thorough analysis of the cryptographic fundamentals of the system as well as improved unit and regression testing for VM correctness. This should provide additional confidence that no single step bias can be propagated across multiple steps to allow unwanted optimizations.

Project Dashboard

Application Summary

Name	RandomX
Type	Proof of work algorithm
Platforms	Cross-Platform (C++)

Engagement Summary

Dates	May 20th to June 3rd 2019
Method	Whitebox
Consultants Engaged	2
Level of Effort	2 person-weeks

Vulnerability Summary

Total High-Severity Issues		
Total Medium-Severity Issues		
Total Low-Severity Issues	2	■■
Total Informational-Severity Issues	1	■
Total Undetermined-Severity Issues		
Total	3	

Category Breakdown

Configuration	1	■
Cryptography	1	■
Data Validation	1	■
Total	3	

Engagement Goals

RandomX is a proof-of-work hashing algorithm primarily developed for use by the Monero blockchain. Its design goal is to be optimized for general purpose CPUs (rather than ASICs or GPU-based execution) while remaining a secure algorithm. To accomplish this, RandomX uses a variety of techniques, including aggressive use of native AES instructions for data transformation, randomized execution on a virtual machine, memory-hard hashing functions, and more. During our analysis we prioritized the following areas:

Cryptography

- Are there methods by which the algorithm could be significantly accelerated?
- Is the algorithm's use of AES rounds safe?
- What is a safe set of parameters to use?

General Security

- Are there inputs that can cause the algorithm to crash?
- Is there undefined behavior in the implementation?
- Is the implementation deterministic and guaranteed to terminate?
- Does the implementation adhere to the RandomX specification?

Coverage

In this section we highlight some of the analysis coverage we achieved during this effort, relative to the areas specified by Arweave in our engagement goals.

Cryptographic Architecture

- The design and specification documentation was reviewed to fully understand the design goals of the algorithm.
- The random program generation code was reviewed to confirm that the generated programs were statistically random.
- The security properties of the AesGenerator and AesHash constructions were scrutinized to determine whether their unusual use of AES was secure.
- Trail of Bits also ran the complete algorithm on many test inputs and examined the results for their randomness properties, as well as examining the randomness within the context of VM execution. A summary of our efforts can be found in Appendix B.

General Security of the Implementation

- Constants (*e.g.*, related to cryptographic algorithms like AES) were compared against their associated specifications.
- Third-party components of the system (*e.g.*, Argon2) were diffed against their latest versions to check for missing upstream bugfixes and security patches.
- The entire codebase was manually inspected for logical errors, undefined behavior, denial of service, and memory-corruption vulnerabilities.

RandomX neither interacts with the network nor the filesystem, nor does it parse complex user input. Therefore, its attack surface is relatively small.

Recommendations Summary

This section aggregates all the recommendations made during the engagement. Short-term recommendations address the immediate causes of issues. Long-term recommendations pertain to the development process and long-term design goals.

Short Term

- ✓ **Increase the number of rounds of AES used in AesGenerator.** AesGenerator uses a single round of AES to create a pseudorandom stream. A minimum of two rounds is needed for diffusion. Insufficient mixing of bits due to single-round operation could allow an attacker, in combination with other biases, to craft inputs that could zero bits in the output. This issue has subsequently been resolved in the code base with a new `AesGenerator4R` that performs 4 rounds. (TOB-ARW-001)
- ❑ **Increase test coverage of the VM semantics and implement regression tests to ensure the consistency of RandomX output.** These should validate that the implementation conforms to the specification. Without such assurances, a cleanroom implementation of the algorithm might deviate from the output of the RandomX reference implementation. Also, future changes to the implementation might cause changes in the VM's behavior and therefore result in a blockchain fork. (TOB-ARW-002)
- ❑ **Add additional comments to dangerous configuration variables.** Several configuration options may weaken the security of the system or compromise its ASIC/GPU resistance. These options should be clearly marked as dangerous to make it easier for other blockchains to adopt RandomX without fear of compromising its security. (TOB-ARW-003)

Long Term

- ❑ **Formally validate the VM semantics against a machine-readable specification.** This will provide additional assurances and protect against consensus issues. (TOB-ARW-002)
- ❑ **Research ways to improve SuperscalarHash's output distribution.** Improvements in this will negate the need for carefully tuned parameters for initial register values and Dataset size, and make the algorithm more resistant to problems if and when other blockchains choose to use it with altered initial values. (Appendix D)
- ❑ **Remove or hide dangerous configuration variables.** Many of RandomX's configurable values are not desirable to change when altering the parameters for a different blockchain. Consider removing them as configuration options entirely or moving their values to a separate header to emphasize the importance of not changing them without thorough investigation. (TOB-ARW-003)

Findings Summary

#	Title	Type	Severity
1	Single AES rounds used in AesGenerator	Cryptography	Low
2	Insufficient Testing and Validation of VM Correctness	Data Validation	Low
3	RandomX configurable parameters are brittle	Configuration	Informational

1. Single AES rounds used in AesGenerator

Severity: Low
Type: Cryptography
Target: `RandomX`

Difficulty: High
Finding ID: TOB-ARW-001

Description
The AES encryptions described by the RandomX specification refer to a single round of AES, namely: ShiftRows, SubBytes, MixColumns, and AddRoundKey. RandomX doesn't depend on the encryption of AES for its security, but rather 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. A single round is insufficient to fully mix.

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

Exploit Scenario
Insufficient mixing of bits due to single round operation could allow an attacker, in combination with other biases, to craft inputs that could zero bits in the output. This would potentially allow deliberate generation of simpler programs to simplify ASIC design or accelerate execution.

Recommendation
Use additional rounds of AES for these transformations to ensure that input diffuses across the entire output range. Increase to a minimum of two, but potentially four if the additional overhead is tolerable. As Joan Daemen and Vincent Rijmen, the authors of AES, wrote in *The Design of Rijndael* (Section 3.5):

> Two rounds of Rijndael provide 'full diffusion' in the following sense: every state bit depends on all state bits two rounds ago, or a change in one state bit is likely to affect half of the state bits after two rounds. Adding four rounds can be seen as adding a 'full diffusion step' at the beginning and at the end of the cipher. The high diffusion of the Rijndael round transformation is thanks to its uniform structure that operates on all state bits.

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. Using four rounds as part of program generation resolves the concerns documented in this issue.

References
- *The Design of Rijndael* by Joan Daemen and Vincent Rijmen, ISBN: 978-3540425809
- `./src/intrin_portable.h#124`

- `./src/soft_aes.cpp#330`

2. Insufficient Testing and Validation of VM Correctness

Severity: Low
Type: Data Validation
Target: `RandomX`

Difficulty: Low
Finding ID: TOB-ARW-002

Description
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. However, this effort was insufficient to validate that the VM implementation is without semantic errors.

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.

Exploit Scenario
A third-party cleanroom implementation of the RandomX specification becomes popular on a blockchain utilizing 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.

Recommendation
Short term, increase test coverage of the VM semantics. Also, implement regression tests to ensure the consistency of RandomX output.

Long term, consider formally validating the VM semantics against a machine-readable specification.

3. RandomX configurable parameters are brittle

Severity: Informational
Type: Configuration
Target: `RandomX`

Difficulty: Undetermined
Finding ID: TOB-ARW-003

Description
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, alternate blockchains interested in using RandomX are required to make different choices for some subset of values due to the threat of 51% attacks. These choices must be made without clear guidance on what knobs should be twisted and which ones may compromise the advantages offered by the algorithm. This brittleness will potentially impede third party adoption.

Exploit Scenario
Another blockchain chooses to utilize RandomX and selects alternate parameters with extremely small values for L1/L2/L3 scratchpad. This choice immediately impacts the algorithm's CPU advantage negatively, potentially negating the resistance to GPU/ASIC that RandomX offers.

Recommendation
Short term, add comments next to the more dangerous configuration variables that advise potential users of what changing them can mean. Each parameter is discussed in greater detail in Appendix E.

Long term, remove or hide more dangerous configuration options entirely.

A. Vulnerability Classifications

Vulnerability Classes	
Class	**Description**
Access Controls	Related to authorization of users and assessment of rights
Auditing and Logging	Related to auditing of actions or logging of problems
Authentication	Related to the identification of users
Configuration	Related to security configurations of servers, devices or software
Cryptography	Related to protecting the privacy or integrity of data
Data Exposure	Related to unintended exposure of sensitive information
Data Validation	Related to improper reliance on the structure or values of data
Denial of Service	Related to causing system failure
Error Reporting	Related to the reporting of error conditions in a secure fashion
Patching	Related to keeping software up to date
Session Management	Related to the identification of authenticated users
Timing	Related to race conditions, locking or order of operations
Undefined Behavior	Related to undefined behavior triggered by the program

Severity Categories	
Severity	**Description**
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or Defense in Depth
Undetermined	The extent of the risk was not determined during this engagement
Low	The risk is relatively small or is not a risk the customer has indicated is important
Medium	Individual user's information is at risk, exploitation would be bad for client's reputation, moderate financial impact, possible legal

	implications for client
High	Large numbers of users, very bad for client's reputation, or serious legal or financial implications

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploit was not determined during this engagement
Low	Commonly exploited, public tools exist or can be scripted that exploit this flaw
Medium	Attackers must write an exploit, or need an in-depth knowledge of a complex system
High	The attacker must have privileged insider access to the system, may need to know extremely complex technical details or must discover other weaknesses in order to exploit this issue

B. Code Quality

The below issues do not pose a direct security risk to RandomX, but can still be security-relevant and should be addressed.

- The `dataset` variable passed to the `randomx_create_vm` function on line 211 of `src/tests/benchmark.cpp` will be uninitialized if `--mine` is not passed as a command line argument. This does not appear to affect the validity of the tests, though, since `dataset` is not used in light verification mode.

- It is generally considered good practice to include braces around every if block, even if the block contains only a single line of code. This practice can help prevent vulnerabilities like Apple's infamous "goto fail" bug.

- The codebase uses namespaces containing `constexpr ints` for constant enumerations. For example, in `instruction.hpp`:

```
namespace InstructionType {
    constexpr int IADD_RS = 0;
    constexpr int IADD_M = 1;
    constexpr int ISUB_R = 2;
    constexpr int ISUB_M = 3;
    constexpr int IMUL_R = 4;
    constexpr int IMUL_M = 5;
    constexpr int IMULH_R = 6;
    constexpr int IMULH_M = 7;
    constexpr int ISMULH_R = 8;
    constexpr int ISMULH_M = 9;
    constexpr int IMUL_RCP = 10;
    constexpr int INEG_R = 11;
    constexpr int IXOR_R = 12;
    constexpr int IXOR_M = 13;
    constexpr int IROR_R = 14;
    constexpr int IROL_R = 15;
    constexpr int ISWAP_R = 16;
    constexpr int FSWAP_R = 17;
    constexpr int FADD_R = 18;
    constexpr int FADD_M = 19;
    constexpr int FSUB_R = 20;
    constexpr int FSUB_M = 21;
    constexpr int FSCAL_R = 22;
    constexpr int FMUL_R = 23;
    constexpr int FDIV_M = 24;
    constexpr int FSQRT_R = 25;
    constexpr int CBRANCH = 26;
    constexpr int CFROUND = 27;
    constexpr int ISTORE = 28;
    constexpr int NOP = 29;
}
```

  While this is preferable to traditional C++ `enums`, RandomX should consider using `enum class` instead, wherever possible. Not only will this retain all of the same compiler optimizations provided by `constexpr int`, but it enables the addition of convenience functions, can catch certain types of bugs related to invalid enum values, and enables enhanced `switch` compiler diagnostics. Alternatively, a standard `enum` can be wrapped in a class instead of a namespace. This will implicitly require `constexpr` values.

C. Randomness Analysis

The final output of RandomX is an invocation of the blake2b hashing function. Blake2 is a widely respected hash function with multiple papers analyzing its security properties as a cryptographic hash. To learn more about the randomness of RandomX we instead look at the source of entropy provided to the final hash to gauge whether or not the behavior prior to blake2b is random.

Our analysis of the generation of initial scratchpads and VM instructions suggests that they are sufficiently random. The execution of the VM will modify the register file and scratchpad according to the randomly generated instructions to produce a random state that is hard to pre-compute. This is then fed into the `AesHash` function, which perturbs the output utilizing repeated single-round AES operations with round keys obtained from the initial state. To ensure more complete diffusion of the bits throughout the output, Trail of Bits recommends increasing the number of rounds (TOB-ARW-001).

Trail of Bits ran statistical tests against AesGenerator by performing over 500 million executions and graphing the output. Figure 1 displays the frequency of byte occurrences with respect to byte value.

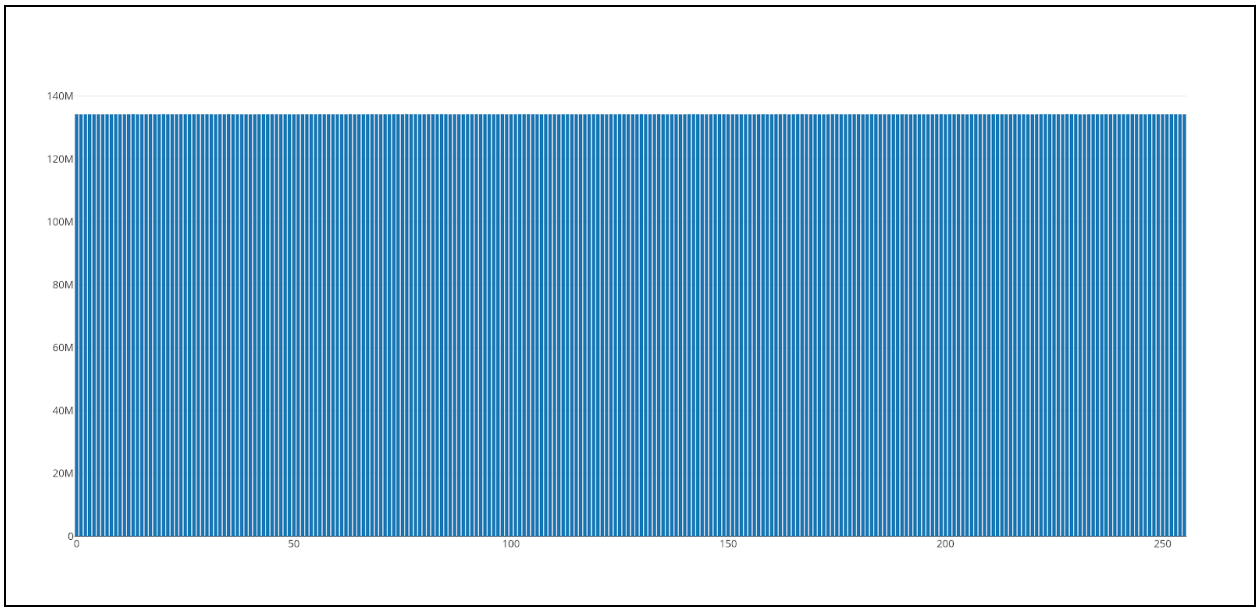

Figure 1: *Byte frequency of AesGenerator given random seeds*

Zooming in to see the variance at the top of the graph (Figure 2) we can see that the maximum variance between the most and least represented set bit is less than 0.02%, with an average near 134.2 million and a deviation of around 23 thousand from the average.

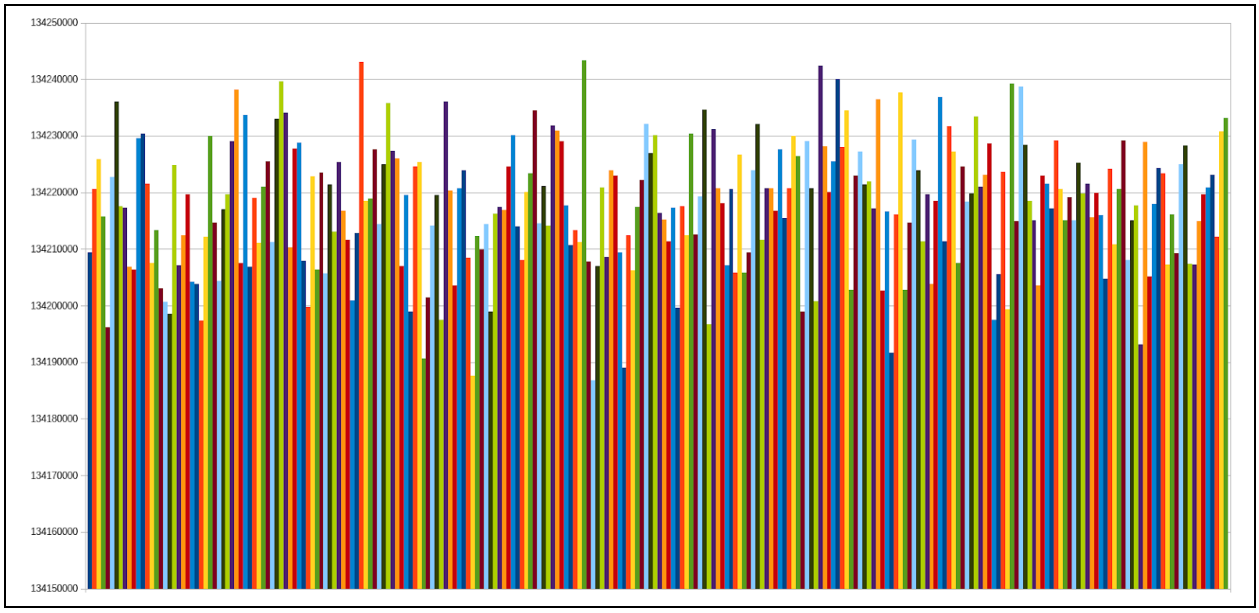

Figure 2: *Zoomed in byte frequency of AesGenerator output*

The distribution of output is within the bounds expected. Note that while checking the distribution is important, a secure PRNG has many other requirements. This algorithm should not be treated as a secure PRNG as the state is easily reversible, but this is not an issue for RandomX since the initial state is publicly known.

Trail of Bits also verified that the default instruction frequency constants matched what was generated in over 500 million executions. The results appear in Figure 3.

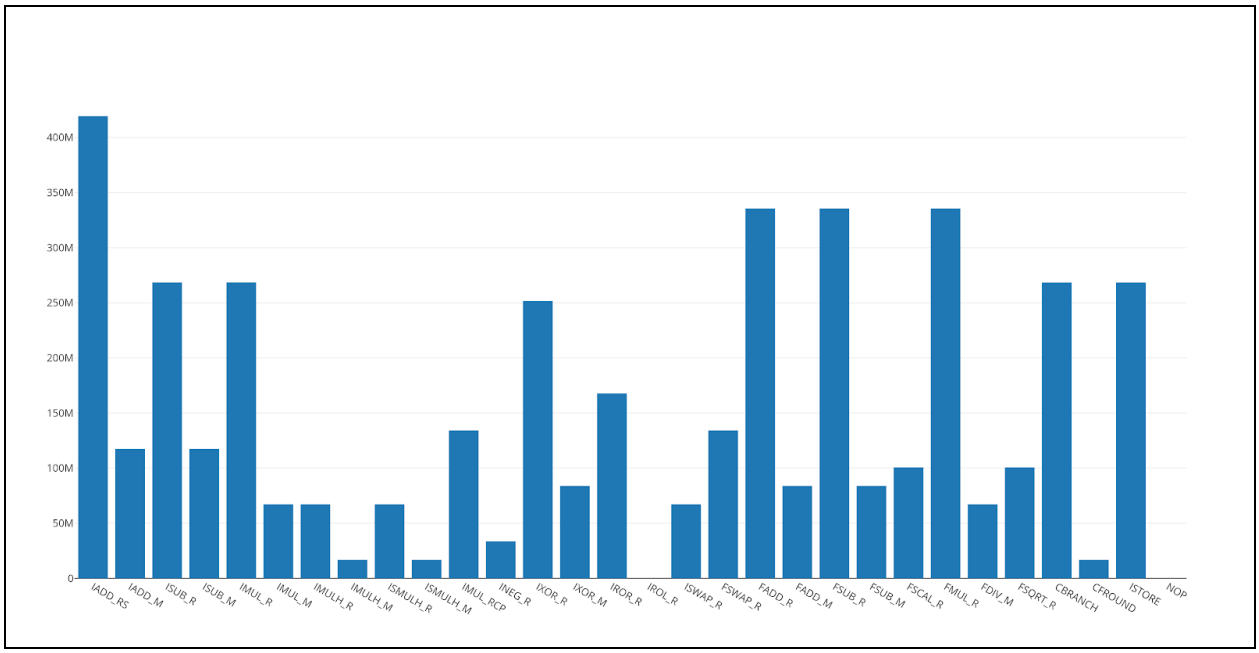

Figure 3: *VM instruction frequencies*

Comparing to the expected values, per 256 instructions, we see the following ratios that continue to converge as additional executions are performed.

Instruction	Expected (Per 256)	Observed	% Difference
IADD_RS	25	24.99956	0.04%
IADD_M	7	7.00021	0.02%
ISUB_R	16	16.00041	0.04%
ISUB_M	7	7.00032	0.03%
IMUL_R	16	16.00131	0.13%
IMUL_M	4	4.00007	0.01%
IMULH_R	4	4.00015	0.01%
IMULH_M	1	0.99996	0.00%
ISMULH_R	4	4.00016	0.02%
ISMULH_M	1	0.99988	0.01%
IMUL_RCP	8	7.99997	0.00%
INEG_R	2	2.00006	0.01%
IXOR_R	15	15.00067	0.07%
IXOR_M	5	4.99930	0.07%
IROR_R	10	9.99988	0.01%
IROL_R	0	0.00000	0.00%
ISWAP_R	4	3.99985	0.02%
FSWAP_R	8	8.00004	0.00%
FADD_R	20	19.99887	0.11%
FADD_M	5	4.99991	0.01%
FSUB_R	20	19.99965	0.04%
FSUB_M	5	4.99970	0.03%
FSCAL_R	6	6.00015	0.01%
FMUL_R	20	20.00016	0.02%
FDIV_M	4	3.99969	0.03%
FSQRT_R	6	5.99905	0.10%

CBRANCH	16	16.00065	0.06%
CFROUND	1	0.99998	0.00%
ISTORE	16	16.00039	0.04%
NOP	0	0.00000	0.00%

D. Bias in SuperscalarHash

SuperscalarHash is a custom diffusion function used as part of the RandomX algorithm. Its primary purpose is to consume as much power as possible in a short period of time by leveraging the superscalar features of a modern general purpose CPU. It takes a 64-byte argument and returns a 64-byte value. Datasets are generated by a process that combines multiple invocations of SuperscalarHash with a variety of other techniques.

To ensure that every key has a high probability of generating a unique Dataset, the SuperscalarHash should ideally exhibit a strong avalanching effect. That is, a single bit change to the input should result in a large change to the output across all bits.

Statistical sampling by the RandomX authors has shown that uniqueness in bits 3–53 of the input registers is required to ensure unique output. This is due to significant output insensitivity at high and low bits. To achieve this, the RandomX algorithm uses a carefully chosen set of initial register values that guarantee unique values for all Dataset item numbers.

The total number of Datasets is

```
RANDOMX_DATASET_BASE_SIZE + RANDOMX_DATASET_EXTRA_SIZE
```

in bytes divided by 64. This, by default, is

2,147,483,648 + 33,554,368 = 2,181,038,016 bytes = 2080 MiB,

which corresponds to 34,078,719 items. The initial register values are tuned for this size, but these parameters are configurable and this potentially makes the construction fragile. This issue is partially mitigated because `RANDOMX_DATASET_BASE_SIZE` has a maximum allowed value of 4,294,967,296 bytes, a value for which the RandomX team also confirmed no collisions, but the only constraint on `RANDOMX_DATASET_EXTRA_SIZE` is that it must be divisible by 64.

An additional mitigating factor for this issue is the number of SuperscalarHash invocations per Dataset item generated. Each Dataset is produced via 8 calls and this significantly increases the probability that a unique output will be generated despite the presence of bias.

Trail of Bits recommends researching ways to improve SuperscalarHash's output distribution such that carefully tuned parameters for both Dataset size and initial register values are not required to obtain the desired result.

E. Parameter Selection

RandomX contains a large set of configurable parameters. The default parameters are well-tuned for the algorithm. However, since Monero already intends to use these values, other blockchains wishing to use RandomX must choose *different* values to prevent 51% attacks. For example, if two blockchains use the same parameterization, then a mining pool from the larger blockchain could maliciously alter the smaller blockchain by overwhelming its own miners.

This section describes each parameter and whether it is safe to change. Many of these parameters interact with each other in subtle ways so care must be taken even when altering values Trail of Bits believes are safe to change.

Parameter	Description	Safe to Change	Notes
RANDOMX_ARGON_MEMORY	The amount of memory the initial argon2d derivation should consume. Used in Cache initialization.	✓ Yes	Increasing this value will increase the minimum memory requirement for light mode.
RANDOMX_ARGON_ITERATIONS	The number of argon2d iterations that should be performed. Used in Cache initialization.	✓ Yes	This controls the time cost of the initial derivation. Higher numbers will take longer.
RANDOMX_ARGON_LANES	The parallelization factor of argon2d. This does not speed up the algorithm, but increases the number of parallel threads executing. Used in Cache initialization.	✓ Yes	This value should be chosen based on the expected hardware the algorithm will run on. Multiple threads makes sense if multiple CPU cores are expected to be present.
RANDOMX_ARGON_SALT	The salt provided to argon2d. Used in cache initialization.	✓ Yes	Any consumer should change this value to a unique string specific to their use.
RANDOMX_CACHE_ACCESSES	The number of Cache accesses per Dataset item generated.	✓ Yes	Higher numbers increase memory bandwidth consumption. The minimum value allowed by the code is 2.
RANDOMX_SUPERSCALAR_LATENCY	This value controls the maximum number of virtual CPU cycles a call to generateSuperscalar will continue to execute. However, in general saturation of the execution ports of the CPU will cause loop termination before this value is reached so it serves as a limit to guarantee loop termination and nothing	✗ No	This value was selected to match the clock speed and DRAM latency of modern computers. Additionally, it potentially interacts with RANDOMX_SUPERSCALAR_MAX_SIZE as a large maximum size may cause this value to be used.

	else.		
RANDOMX_SUPERSCALAR_MAX_SIZE	The maximum number of instructions a SuperscalarHash program can execute.	✓ **Yes, but...**	Do not set this value lower than the default (512).
RANDOMX_DATASET_BASE_SIZE	The base size of the Dataset. This value must be a power of 2 and cannot be more than 4,294,967,296 bytes.	✓ **Yes, but...**	Keep this value as high as your usage allows. This affects both the behavior of SuperscalarHash (Appendix D) as well as the light versus fast mode memory ratios.
RANDOMX_DATASET_EXTRA_SIZE	Additional size to store more Dataset items. This value must be divisible by 64 but is otherwise unbounded.	✓ **Yes, but...**	Keep this value very small compared to RANDOMX_DATASET_BASE_SIZE. It should be used only as a value to slightly increase the total memory consumed. For example, if base size is 1GiB this might appropriately be set to approximately 16MiB. This value affects both the behavior of SuperscalarHash (Appendix D) as well as the light versus fast mode memory ratios.
RANDOMX_PROGRAM_SIZE	The number of instructions in a RandomX program executed on the virtual machine.	✓ **Yes, but...**	Do not set this value lower than the default (256). Significantly larger values may also fail to fit in a CPU's L1 instruction cache, resulting in slower than expected operation.
RANDOMX_PROGRAM_ITERATIONS	The number of times a program is looped during virtual machine execution.	✓ **Yes, but...**	Do not set this value lower than the default (2048). Programs perform memory writes, so lower values may affect randomness of intermediate steps.
RANDOMX_PROGRAM_COUNT	The number of chained virtual machine executions per RandomX hash.	✓ **Yes, but...**	Do not set this value lower than the default (8). Chained execution is a means of preventing attackers from gaining an advantage by only executing "easy" programs.
RANDOMX_JUMP_BITS	The number of consecutive bits that must be zero to execute the jump.	✓ **Yes, but...**	The performance characteristics of RandomX include deliberate low probability branching to provide advantage in speculation execution so modification of this value should be small. A higher value leads to less jumps, a lower value leads to more jumps. Setting this to 10 would cause a branch in 1 in 1024 (2^{10}) where the default of 8 branches 1 in 256 (2^8). The sum of RANDOMX_JUMP_BITS and RANDOMX_JUMP_OFFSET must be less than or equal to 16.
RANDOMX_JUMP_OFFSET	The offset to apply to determine changes to the jump condition.	✓ **Yes, but...**	The value helps calculate an offset that is used to modify bits in a random value. The sum of RANDOMX_JUMP_BITS and RANDOMX_JUMP_OFFSET must be less

			than or equal to 16.
RANDOMX_SCRATCHPAD_L3	The RandomX virtual machine's level 3 cache size.	✗ No	The cache sizes for the virtual machine were chosen to provide specific performance characteristics with regard to the way a general purpose CPU will cache the data. Alteration of these values is not advised.
RANDOMX_SCRATCHPAD_L2	The RandomX virtual machine's level 2 cache size.	✗ No	The cache sizes for the virtual machine were chosen to provide specific performance characteristics with regard to the way a general purpose CPU will cache the data. Alteration of these values is not advised.
RANDOMX_SCRATCHPAD_L1	The RandomX virtual machine's level 1 cache size.	✗ No	The cache sizes for the virtual machine were chosen to provide specific performance characteristics with regard to the way a general purpose CPU will cache the data. Alteration of these values is not advised.

Instruction Frequency

In addition to these defaults, RandomX also allows alteration of the frequency of instructions in programs generated for the virtual machine. These defaults are shown in Figure 4.

```
/*
Instruction frequencies (per 256 opcodes)
Total sum of frequencies must be 256
*/
#define RANDOMX_FREQ_IADD_RS       25
#define RANDOMX_FREQ_IADD_M         7
#define RANDOMX_FREQ_ISUB_R        16
#define RANDOMX_FREQ_ISUB_M         7
#define RANDOMX_FREQ_IMUL_R        16
#define RANDOMX_FREQ_IMUL_M         4
#define RANDOMX_FREQ_IMULH_R        4
#define RANDOMX_FREQ_IMULH_M        1
#define RANDOMX_FREQ_ISMULH_R       4
#define RANDOMX_FREQ_ISMULH_M       1
#define RANDOMX_FREQ_IMUL_RCP       8
#define RANDOMX_FREQ_INEG_R         2
#define RANDOMX_FREQ_IXOR_R        15
#define RANDOMX_FREQ_IXOR_M         5
#define RANDOMX_FREQ_IROR_R        10
#define RANDOMX_FREQ_IROL_R         0
#define RANDOMX_FREQ_ISWAP_R        4
#define RANDOMX_FREQ_FSWAP_R        8
#define RANDOMX_FREQ_FADD_R        20
#define RANDOMX_FREQ_FADD_M         5
#define RANDOMX_FREQ_FSUB_R        20
#define RANDOMX_FREQ_FSUB_M         5
#define RANDOMX_FREQ_FSCAL_R        6
#define RANDOMX_FREQ_FMUL_R        20
#define RANDOMX_FREQ_FDIV_M         4
#define RANDOMX_FREQ_FSQRT_R        6
#define RANDOMX_FREQ_CBRANCH       16
#define RANDOMX_FREQ_CFROUND        1
#define RANDOMX_FREQ_ISTORE        16
#define RANDOMX_FREQ_NOP            0
```

Figure 4: *Instruction frequency*

The default frequencies were chosen to provide a mix of operations, including memory writes and reads, that strongly benefit modern superscalar, speculative, and out-of-order execution. Trail of Bits does not recommend changing these frequencies without conducting significant statistical analyses to confirm that any altered values preserve the security, performance, and ASIC/GPU resistance advantages.

F. Recommended Parameters for Arweave

The cryptographic analysis conducted in this report is predicated on the default values, especially related to our statistical analyses of randomness. To preserve the properties observed while breaking compatibility with other blockchains using the same algorithm, Trail of Bits suggests that Arweave alter the following values:

Parameter	Default Value	Suggested Value
RANDOMX_ARGON_ITERATIONS	3	5
RANDOMX_ARGON_LANES	1	2
RANDOMX_ARGON_SALT	"RandomX\x03"	"RandomX-Arweave\x01"
RANDOMX_PROGRAM_SIZE	256	248
RANDOMX_PROGRAM_COUNT	8	9

Table 1: *Alternate Parameter Recommendations*

These suggested parameters are conservative choices that should provide security equivalent to the default parameters while ensuring that systems built expecting the default parameters will not trivially interoperate. The values themselves are not special beyond a slight increase in some places (and a slight decrease in one) to ensure the algorithm's assumptions are not weakened while preserving similar performance. However, making certain that they differ from the default is critical to avoid potential 51% attacks by other blockchains that utilize the same parameters.

Parameters related to the virtual CPU (such as scratchpad size) can significantly affect performance, as these defaults were carefully chosen to match the characteristics of existing general-purpose CPUs. They should not be changed.