Statistical Model Checking of RANDAO's Resilience to Pre-computed Reveal Strategies

Musab A. Alturki[1,2] and Grigore Roşu[3,1]

[1] Runtime Verification Inc., Urbana, IL 61801
[2] King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia
musab.alturki@kfupm.edu.sa
[3] University of Illinois at Urbana-Champaign, Urbana, IL 61801
grosu@illinois.edu

Abstract. RANDAO is a commit-reveal scheme for generating pseudo-random numbers in a decentralized fashion. The scheme is used in emerging blockchain systems as it is widely believed to provide randomness that is unpredictable and hard to manipulate by maliciously behaving nodes. However, RANDAO may still be susceptible to look-ahead attacks, in which an attacker (controlling a subset of nodes in the network) may attempt to pre-compute the outcomes of (possibly many) reveal strategies, and thus may bias the generated random number to his advantage. In this work, we formally evaluate resilience of RANDAO against such attacks. We first develop a probabilistic model in rewriting logic of RANDAO, and then apply statistical model checking and quantitative verification algorithms (using Maude and PVeStA) to analyze two different properties that provide different measures of bias that the attacker could potentially achieve using pre-computed strategies. We show through this analysis that unless the attacker is already controlling a sizable percentage of nodes while aggressively attempting to maximize control of the nodes selected to participate in the process, the expected achievable bias is quite limited.

Keywords: RANDAO · Rewriting logic · Maude · Statistical Model Checking · Blockchain

1 Introduction

Decentralized pseudo-random value generation is a process in which participants in a network, who do not necessarily trust each other, collaborate to produce a random value that is unpredictable to any individual participant. It is a core process of many emerging distributed autonomous systems, most prominently proof-of-stake (PoS) consensus protocols, which include the upcoming Ethereum 2.0 (a.k.a. Serenity) protocol [11, 8]. A commonly accepted implementation scheme for decentralized random value generation is a commit-reveal scheme, known as RANDAO (due to Youcai Qian [16]), in which participants first make commitments by sharing hash values of seeds, and then, at a later stage, they reveal their seeds, which can then be used for generating the random value. In a PoS

protocol, and in particular in Serenity [11], the scheme is used repeatedly in a sequence of rounds in such a way that the outcome of a round is used as a seed for generating the random value of the following round. Moreover, the scheme is usually coupled with a reward system that incentivizes successful participation and discourages deviations from the protocol. Several other distributed protocols have also adopted this scheme primarily for its simplicity and flexibility.

However, this approach may still be susceptible to *look-ahead* attacks, in which a malicious participant may choose to refrain from revealing his seed if skipping results in randomness that is more favorable to him. In general, a powerful attacker may attempt to pre-compute the outcomes of (possibly many) reveal strategies, which are sequences of reveal-or-skip decisions, and thus may anticipate the effects of his contribution to the process and bias the generated random number to his advantage.

While this potential vulnerability is known and has been pointed to in several works in the literature (e.g. [7, 6, 4]), the extent to which it may be exploited by an attacker and how effective the attack could be in an actual system, such as a PoS system like Serenity, have not yet been thoroughly investigated, besides the exploitability arguments made in [7] and [6], which were based on abstract analytical models. While the high-level analysis given there is useful for gaining a foundational understanding of the vulnerability and the potential of the attack, a lower-level formalization that captures the interactions of the different components of the RANDAO process and the environment could provide deeper insights into how realizable the attack is in an actual system.

In this work, we develop a computational model of the RANDAO scheme as a probabilisitic rewrite theory [12, 1] in rewriting logic [13] to formally evaluate resilience of RANDAO to pre-computed reveal strategies. The model gives a formal, yet natural, description of (possibly different designs of) the RANDAO process and the environment. Furthermore, the model is both *timed*, capturing timing of events in the process, and *probabilistic*, modeling randomized protocol behaviors and environment uncertainties. Being executable, the model facilitates automated formal analysis of *quantitative properties*, specified as real-valued formulas in QuaTEx (Quantitative Temporal Expressions Logic) [1], through efficient statistical model checking and quantitative analysis algorithms using both Maude [9] (a high-performance rewriting system) and PVeStA [2] (a statistical verification tool that interfaces with Maude). Using the model, we analyze two properties that provide different measures of bias that the attacker could potentially achieve using pre-computed strategies: (1) the *matching score*, which is the expected number of proposers that the attacker controls, and (2) the *last-word score*, which is the length of the longest tail of the proposers list that the attacker controls.

We show through this analysis that unless the attacker is already controlling a sizable portion of validators and is aggressively attempting to maximize the number of last compromised proposers in the proposers list, or what we call the *compromised tail* of the list, the expected achievable bias of randomness of the RANDAO scheme is quite limited. However, an aggressive attacker who can

afford to make repeated skips for very extended periods of time (e.g. in thousands of rounds), or an attacker who already controls a fairly large percentage (e.g. more than 30%) of participants in the network will have higher chances of success.

The rest of the paper is organized as follows. In Section 2, we quickly review rewriting logic and statistical model checking. In Section 3, we introduce in some detail the RANDAO scheme. This is followed in Section 4 by a description of our model of RANDAO in rewriting logic. Section 5 the analysis properties and results. The paper concludes with a discussion of future work in Section 6.

2 Background

Rewriting logic [14] is a general logical and semantic framework in which systems can be formally specified and analyzed. A unit of specification in rewriting logic is a *rewrite theory* $\mathcal{R}$, which formally describes a concurrent system including its static structure and dynamic behavior. It is a tuple $(\Sigma, E \cup A, R)$ consisting of: (1) a membership equational logic (MEL) [15] signature Σ that declares the kinds, sorts and operators to be used in the specification; (2) a set E of Σ-sentences, which are universally quantified Horn clauses with atoms that are either equations $(t = t')$ or memberships $(t : s)$; (3) A a set of equational axioms, such as commutativity, associativity and/or identity axioms; and (4) a set R of rewrite rules $t \longrightarrow t'$ if C specifying the computational behavior of the system (where C is a conjunction of equational or rewrite conditions). Operationally, if there exists a substitution θ such that $\theta(t)$ matches a subterm s in the state of the system, and $\theta(C)$ is satisfied, then s may rewrite to $\theta(t')$. While the MEL sub-theory $(\Sigma, E \cup A)$ specifies the user-defined syntax and equational axioms defining the system's state structure, a rewrite rule in R specifies a *parametric transition*, where each instantiation of the rule's variables that satisfies its conditions yields an actual transition (See [5] for a detailed account of generalized rewrite theories).

Probabilistic rewrite theories extend regular rewrite theories with probabilistic rules [17, 1]. A probabilistic rule ($t \longrightarrow t'$ if C with probability π) specifies a transition that can be taken with a probability that may depend on a probability distribution function π parametrized by a t-matching substitution satisfying C. Probabilistic rewrite theories unify many different probabilistic models and can express systems involving both probabilistic and nondeterministic features.

Maude [9] is a high-performance rewriting logic implementation. An equational theory $(\Sigma, E \cup A)$ is specified in Maude as a functional module, which may consist of sort and subsort declarations for defining type hierarchies, operator declarations, and unconditional and conditional equations and memberships. Operator declarations specify the operator's syntax (in mixfix notation), the number and sorts of the arguments and the sort of its resulting expression. Furthermore, equational attributes such as associativity and commutativity axioms may be specified in brackets after declaring the input and output sorts. A rewrite theory is specified as a *system module*, which may additionally contain rewrite rules declared with the `rl` keyword (`crl` for conditional rules).

Furthermore, probabilistic rewrite theories, specified as system modules in Maude [9], can be simulated by sampling from probability distributions. Using PVeStA [2], randomized simulations generated in this fashion can be used to statistically model check quantitative properties of the system. These properties are specified in a rich, quantitative temporal logic, QuaTEx [1], in which real-valued state and path functions are used instead of boolean state and path predicates to quantitatively specify properties about probabilistic models. QuaTEx supports parameterized recursive function declarations, a standard conditional construct, and a *next* modal operator $\bigcirc$, allowing for an expressive language for real-valued temporal properties (Example QuaTEx expressions appear in Section 5). Given a QuaTEx path expression and a Maude module specifying a probabilistic rewrite theory, statistical quantitative analysis is performed by estimating the *expected value* of the path expression against computation paths obtained by Monte Carlo simulations. More details can be found in [1].

3 The RANDAO Scheme

The RANDAO scheme [16] is a commit-reveal scheme consisting of two stages: (1) the commit stage, in which a participant p_i first commits to a seed s_i (by announcing the hash of the seed h_{s_i}), and then (2) the reveal stage, in which the participant p_i reveals the seed s_i. The sequence of revealed seeds $s_0, s_1, \cdots, s_{n-1}$ (assuming n participants) are then used to compute a new seed s (e.g. by taking the XOR of all s_i), which is then used to generate a random number.

In the context of the Serenity protocol [11], the RANDAO scheme proceeds in rounds corresponding to epochs in the protocol. At the start of an epoch i, the random number r_{i-1} generated in the previous round (in epoch $i-1$) is used for sampling from a large set of validators participating in the protocol an ordered list of block proposers $p_0, p_1, \cdots p_{k-1}$, where k is the cycle length of the protocol (a fixed number of time slots constituting one epoch in the protocol). Each proposer p_i is assigned the time slot i of the current round (epoch). During time slot i, the proposer p_i is expected to submit the pair (c_{p_i}, s_{p_i}), with c_{p_i} a commitment on a seed to be used for the next participation in the game (in some future round when p_i is selected again as a proposer), and s_{p_i} the seed to which p_i had previously committed in the last participation in the game (or when p_i first joined the protocol's validator set). The RANDAO contract keeps track of successful reveals in the game, which are those reveals that arrive in time and that pass the commitment verification step. Towards the end of an epoch i, the RANDAO contract combines the revealed seeds in this round by computing their XOR s_i, which is used as the seed for the next random number r_{i+1} to be used in the next round $i+1$. To discourage deviations from the protocol and encourage proper participation, the RANDAO contract penalizes proposers who did not successfully reveal (by discounting their Ether deposits) and rewards those proposers who have been able to successfully reveal their seeds (by distributing dividends in Ether).

4 A Rewriting Model of RANDAO

We use rewriting logic [14], and its probabilistic extensions [12, 1], to build a generic and executable model of the RANDAO scheme. The model is specified as a probabilistic rewrite theory $\mathcal{R} = (\Sigma_{\mathcal{R}}, E_{\mathcal{R}} \cup A_{\mathcal{R}}, R_{\mathcal{R}})$, implemented in MAUDE as a system module. By utilizing different facilities provided by its underlying formalism, the model $\mathcal{R}$ is both (purely) *probabilistic*, specifying randomized behaviors and environment uncertainties, and *real-time*, capturing (dense) time clocks and message transmission delays. Furthermore, the model is *parametric* to a number of parameters, such as the attack probability, the size of the validator set and the network latency, to enable capturing different attack scenarios.

In this section, we describe generally the most fundamental parts of the model. A more detailed description of the model can be found in [3].

4.1 Protocol State Structure

The structure of the model, specified by the MEL sub-theory $(\Sigma_{\mathcal{R}}, E_{\mathcal{R}} \cup A_{\mathcal{R}})$ of $\mathcal{R}$, is based on a representation of actors in rewriting logic, which builds on its underlying object-based modeling facilities. In this model, the state of the protocol is a *configuration* consisting of a multiset of actor objects and messages in transit. Objects communicate asynchronously by message passing. An object is a term of the form `<name: O | A >`, with `O` the actor object's unique name (of the sort `ActorName`) and `A` its set of attributes, constructed by an associative and commutative comma operator `_,_` (with `mt` as its identity element). Each attribute is a name-value pair of the form `attr : value`. A message destined for object `O` with payload `C` is represented by a term of the form `O <- C`, where the payload `C` is a term of the sort `Content`.

Objects The three most important objects in the model are: (1) the blockchain object, (2) the RANDAO contract object, and (3) the attacker object.

The blockchain object. This object, identified by the actor name operator `bc`, models abstractly the public data maintained in a blockchain:

```
<name: bc | vapproved: VHL,  vapproved-size: N,
            vpending:  VHL', vpending-size:  N',
            seed: S >
```

The object maintains a list of validator records of all approved and participating validators in the system in an attribute `vapproved`, with its current length in the `vapproved-size` attribute. As new validators arrive and request to join the system, the blockchain object accumulates these incoming requests as a growing list of validator records in its attribute `vpending`, along with its current size in the attribute `vpending-size`. Finally, this object maintains the seed value that was last computed by the previous round of the game in its `seed` attribute.

The RANDAO object. This object, identified by the operator `r`, models a RANDAO contract managing the RANDAO process:

```
<name: r | status:    U, balance:    N,  precords: PL,
           prop-size: M, prop-ilist: IL, pnext: I >
```

It maintains a `status` attribute, indicating its current state of processing, and a `balance` attribute, keeping track of the total contract balance. Moreover, the object manages the proposers list for the current round of the game using the attributes `prop-ilist`, a list of indices identifying the proposers, and `precords`, a list of proposer records of the form `[v(I), B]` with `B` a Boolean flag indicating whether the proposer `v(I)` has successfully revealed. Additionally, the size of the proposers list is stored in `prop-size`. Finally, the object also keeps track of the next time slot (in the current round) to be processed in the attribute `pnext`.

The attacker object. The attacker is modeled by the attacker object, identified by the operator `a`:

```
<name: a | vcomp:    CVL, vcomp-ilist: IL, vcomp-size: N,
           strategy: G >
```

The full list of the compromised validator indices is maintained by the attacker object in the attribute `vcomp-ilist`. This list is always a sublist of the active validators maintained by the blockchain object above. Its length is maintained in the attribute `vcomp-size`. Since in every round of the game, a portion of validators selected as proposers may be compromised, the attacker object creates compromised validator records for all such validators to assign them roles for the round and maintains these records in its attribute `vcomp`. If any one of these compromised validators is at the head of the longest compromised tail of the proposers list, the computed reveal strategy (whenever it becomes ready during the current round) is recorded in the attribute `strategy`.

The Scheduler In addition to objects and messages, the state (configuration) includes a *scheduler*, which is responsible for managing the time domain, modeled by the real numbers, and the scheduling of message delivery. The scheduler is a term of the form `{T | L}`, with `T` the current global clock value and `L` a time-ordered list of scheduled messages, where each such message is of the form `[T,M]`, representing a message `M` scheduled for processing at time `T`. As time advances, scheduled messages in `L` are delivered (in time-order) to their target objects, and newly produced messages by objects are appropriately scheduled into `L`. The scheduler is key in ensuring absence of any unquantified non-determinism in the model, which is a necessary condition for soundness of statistical analysis [3].

4.2 Protocol Transitions

The protocol's state transitions are modeled using the (possibly conditional and/or probabilistic) rewrite rules $R_{\mathcal{R}}$ of the rewrite theory $\mathcal{R} = (\Sigma_{\mathcal{R}}, E_{\mathcal{R}} \cup$

$A_{\mathcal{R}}, R_{\mathcal{R}}$). The rules specify: (1) the actions of the RANDAO contract, which are advancing the time slot, advancing the round and processing validator reveals, and (2) the behaviors of both honest and compromised validators. For space consideration, we only list and describe the rule for advancing the time slot below, while omitting some of the details. Complete descriptions of all the rules can be found in the extended report [3].

The transition for advancing the time slot specifies the mechanism with which the RANDAO contract object checks if a successful reveal was made by the proposer assigned for the current time slot:

```
rl [RAdvanceSlot] :
<name: bc | vapproved-size: N, vpending-size: N',
            seed: S, AS >
<name: r | status: ready, precords: ([ VID , B ] ; CL),
            prop-ilist: IL, pnext: K, AS' >
{ TG | SL } (RID <- nextSlot(L)) ...
=>
<name: bc | vapproved-size: N, vpending-size: N',
            seed: S, AS >
if L > #CYCLE-LENGTH then
  <name: r | status: processing,
             precords: ([ VID , B ] ; CL),
             prop-ilist:
            sampleIndexList(N + N', #CYCLE-LENGTH, S, nilIL),
             pnext: 1, AS' >
  { TG | SL } (RID <- nextRound)
else
  if L == K then
    <name: r | status: ready,
               precords: ([ VID , B ] ; CL),
               prop-ilist: IL, pnext: K, AS' >
  else
    <name: r | status: ready,
               precords: (CL ; [ VID , false ]),
               prop-ilist: IL, pnext: s(K), AS' >
  fi
  insert({ TG | SL }, [TG + 1.0, (RID <- nextSlot(s(L)))])
fi ... .
```

When the current time slot `L` is about to end, the message `nextSlot(L)` becomes ready for the RANDAO object to consume, which initiates the process of advancing the state of the protocol to the next slot. There are three cases that need to be considered depending on the value of `L`:

1. `L > #CYCLE-LENGTH`, meaning that the message's time slot number exceeds the number of slots in a round (slot numbering begins at 1), and thus, the protocol has already processed all slots of the current round, and progressing to the next slot would require advancing the the current round of the game first. Therefore, the RANDAO contract object changes its status to

`processing` and samples a new list of proposers for the next round using the seed `S` that was computed in the current round. The object resets the time slot count to 1 and emits a self-addressed, zero-delay `nextRound` message.
2. `L == K`, where `K` is the next-slot number stored in the RANDAO object, which means that the slot number `K` was already advanced by successfully processing a reveal some time earlier during this slot's time window. In this case, the state is not changed and a `nextSlot(s(L))` message (with `s` the successor function) is scheduled to repeat this process for the next time slot.
3. Otherwise, the slot number `K` stored in the object has not been advanced before and, thus, either a reveal for the current time slot `L` was attempted and failed or that a reveal was never received. In both cases, the RANDAO object records that as a failure in the proposers record list, advances the slot number `K` and schedules a `nextSlot(s(L))` message in preparation for the next time slot.

These cases are specified by the nested conditional structure shown in the rule.

5 Statistical Verification

We use the model $\mathcal{R}$ to formally and quantitatively evaluate how much an attacker can bias randomness of the RANDAO process assuming various attacker models and protocol parameters. In the analysis presented below, we assume a 95% confidence interval with size at most 0.02. We also assume no message drops and random message transmission delays in the range [0.0, 0.1] time units (so reveals, if made, are guaranteed to arrive on time).

5.1 Matching Score (MS)

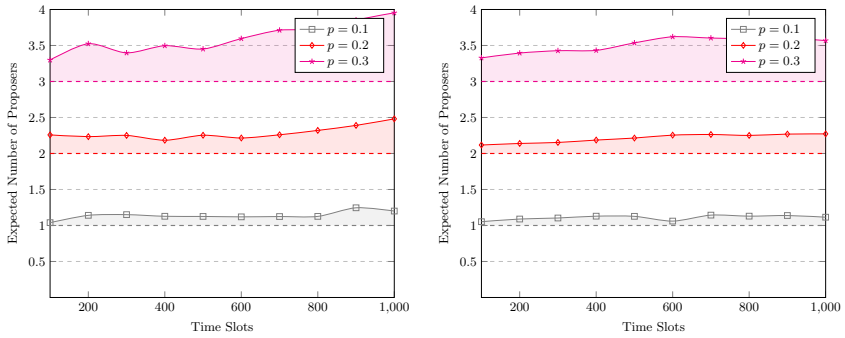
The *Matching score* (MS) is the number of attacker-controlled validators selected as proposers in a round of the RANDAO process. The baseline value for MS (assuming no attack) is given by the expectation of a binomial random variable X with success probability p (the probability of a validator being compromised) in k repeated trials (k is the length of the proposers list), which is:

$$EX[X] = kp \tag{1}$$

As a *temporal* formula in QuaTEx, the property MS is expressed as:

$$\begin{aligned} ms(t) = \ & \underline{\textbf{if}}\ time() > t\ \underline{\textbf{then}}\ countCompromised() \\ & \underline{\textbf{else}}\ \bigcirc\, ms(t)\ \underline{\textbf{fi}}\ ; \\ & \underline{\textbf{eval}}\ \mathbf{E}[ms(t_0)] \end{aligned} \tag{2}$$

$ms(t)$ is a recursively defined path expression that uses two state functions: (1) $time()$, which evaluates to the time value of the current state of the protocol (given by the scheduler object), and (2) $countCompromised()$, which evaluates



(a) MS (10x500) (b) MS (10x1000)

Fig. 1. The expected number of attacker-controlled proposers in the proposers list against execution time in time slots, assuming the attacker is attempting to maximize the number of compromised proposers. The dashed lines represent the base values (with no active attack) computed using Equation (1). The shaded areas visualize the expected bias achievable by the attacker for the three different attack probabilities plotted. We assume a proposers list of size 10, and a validator set of size (a) 10×500 and (b) 10×1000.

to the number of compromised proposers in the current state of the RANDAO object. Therefore, given an execution path, the path expression $ms(t)$ evaluates to *countCompromised*() in the current state if the protocol run is complete (reached the time limit); otherwise, it returns the result of evaluating itself in the next state, denoted by the next-state temporal operator $\bigcirc$. The number of compromised proposers that an attacker achieves (on average) within the time limit specified can be approximated by estimating the expected value of the formula over random runs of the protocol, denoted by the query **eval** $\mathbf{E}[ms(t_0)]$.

The analysis results for MS are plotted in the charts of Figure 1. We use the notation $a \times b$ to denote the fact that the length of the proposers list (`CYCLE-LENGTH`) is a and that there are a total of $a \times b$ participating validators in the configuration[4]. The dashed lines in the charts represent the base values (with no active attack) computed using Equation (1) for different attack probabilities p, while the plotted data points are the model's estimates.

As the charts show, the attacker can reliably but minimally bias randomness with this strategy. This, however, assumes that the attacker is able to afford all the skips that will have to be made in the process, since only after about 80 rounds or so, the attacker is able to gain an advantage of about 20% (over the

[4] The specific values for a and b used in this section and Section 5.2 are chosen so that the total size of the validator set $a \cdot b$ is large enough relative to the length of the proposers list a so that the probability of picking a compromised proposer stays the same (recall that the attack probability is fixed), while not too large to allow efficient analysis. This has the important consequence that the analysis results obtained are representative of actual setups (where the set of validators is much larger than that of the proposers), regardless of the exact proportion of proposers to validators.

baseline). Nevertheless, an attacker that already controls a significant portion of the validators can capitalize on that to speed up his gains, as can be seen from the $p = 0.3$ attacker at around 100 rounds, compared with the weaker attackers. Furthermore, by comparing the charts in Figure 1, we note that results obtained for different proportions of proposers to validators are generally similar.

5.2 Last-Word Score (LWS)

This is the length of the longest attacker-controlled tail of the proposers list in a round of the RANDAO process. We first compute a baseline value for LWS (assuming no attack). Let a be the event of picking an attacker-controlled validator, which has probability p, and b the event of picking an honest validator b, having probability $(1 - p)$. Let the length of the proposers list be k. A compromised tail in the proposers list corresponds to either a sequence of events a of length $j < k$ followed immediately by exactly one occurrence of event b, or a sequence of events a of length exactly k (the whole list is controlled by the attacker). Therefore, letting X be a random variable corresponding to the length of the longest compromised tail, we have:

$$Pr[X = i] = \begin{cases} p^i(1-p) & i < k \\ p^i & i = k \end{cases}$$

Therefore, the expected value of X is

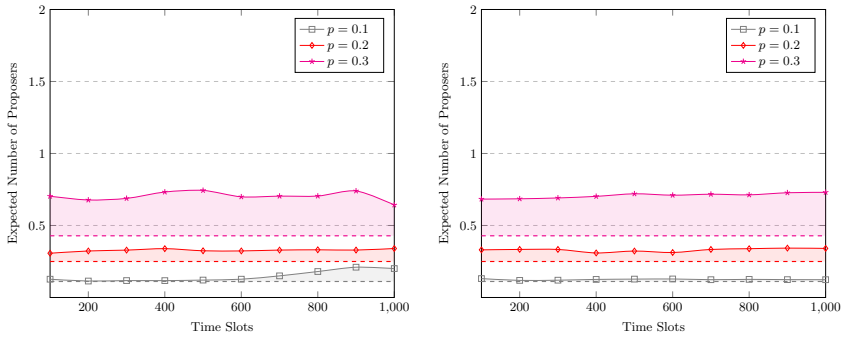
$$EX[X] = \sum_{i=0}^{k-1} i \cdot p^i(1-p) + k \cdot p^k \tag{3}$$

We then specify the property LWS using the following formula:

$$\begin{aligned} lws(t) = {} & \underline{\textbf{if}}\ time() > t\ \underline{\textbf{then}}\ countCompromisedTail() \\ & \underline{\textbf{else}}\ \bigcirc\ lws(t)\ \underline{\textbf{fi}}\ ; \\ & \underline{\textbf{eval}}\ \mathbf{E}[lws(t_0)] \end{aligned} \tag{4}$$

The formula uses the state function *countCompromisedTail*(), which counts the number of proposers in the longest compromised tail in the proposers list of the current state of the RANDAO object. As before, estimating the expectation expression $\mathbf{E}[lws(t_0)]$ gives an approximation of the expected length of the longest compromised tail that an attacker can achieve within the specified time limit.

The results are plotted in the charts of Figure 2. As Figure 2 shows, maximizing the length of the compromised tail can result in a steady and reliable effect on the proposers list. As the attack probability increases, the bias achieved can be greater within shorter periods of time. For example, at around 60 rounds, the bias achieved by a 0.1 attacker is negligible, while a 0.2 attacker is expected to achieve 20% gains over the baseline (at around 0.32 compared with 0.25), and a 0.3 attacker achieves 60% gains (at around 0.7 compared with 0.43). Nevertheless, even at high attack rates, the charts do not show strong increasing trends, suggesting that any gains more significant than those would require applying reveal strategies for very extended periods of time.



(a) LWS (10x500) (b) LWS (10x1000)

Fig. 2. The expected number of attacker-controlled proposers in the proposers list against execution time in time slots, assuming the attacker is attempting to maximize the length of the compromised tail. The dashed lines represent the base values (with no active attack) computed using Equation (3). The shaded areas visualize the expected bias achievable by the attacker for the three different attack probabilities plotted. We assume a proposers list of size 10, and a validator set of size (a) 10×500 and (b) 10×1000.

6 Conclusion

We presented an executable formalization of the commit-reveal RANDAO scheme as a probabilistic rewrite theory in rewriting logic. Through its specification in Maude, we used the model to analyze resilience of RANDAO against pre-computed reveal strategies by defining two quantitative measures of achievable bias: the matching score (MS) and the last-word score (LWS), specified as temporal properties in QuaTEx and analyzed using statistical model checking and quantitative analysis with PVeStA. Further analysis could consider other scenarios with dynamic validator sets, unreliable communication media and extended network latency. Furthermore, the analysis presented does not explicitly quantify the costs to the attacker, which can be an important economic defense against mounting these reveal strategies. An extension of the model could keep track of the number of skips, or specify a limit on these skips, so that the success of an attack strategy can be made relative to the cost of executing it. Finally, a holistic approach to analyzing quantitative properties of Serenity looking into availability and attack resilience properties makes for an interesting longer-term research direction.

Acknowledgements. We thank Danny Ryan and Justin Drake from the Ethereum Foundation for their very helpful comments. This work was performed under the first Ethereum Foundation security grant "Casper formal verification"[10].

References

1. Agha, G., Meseguer, J., Sen, K.: PMaude: Rewrite-based specification language for probabilistic object systems. Electronic Notes in Theoretical Computer Science **153**(2), 213–239 (2006)
2. Alturki, M.A., Meseguer, J.: PVeStA: A parallel statistical model checking and quantitative analysis tool. In: Corradini, A., Klin, B., Cîrstea, C. (eds.) Algebra and Coalgebra in Computer Science, Lecture Notes in Computer Science, vol. 6859, pp. 386–392. Springer Berlin / Heidelberg (2011)
3. Alturki, M.A., Roşu, G.: Statistical model checking of randao's resilience against pre-computed reveal strategies. Tech. rep., The University of Illinois at Urbana-Champaign, http://hdl.handle.net/2142/102076 (November 2018)
4. Boneh, D., Bonneau, J., Bünz, B., Fisch, B.: Verifiable delay functions. In: Proceedings of Crypto 2018. pp. 757–788 (2018)
5. Bruni, R., Meseguer, J.: Semantic foundations for generalized rewrite theories. Theor. Comput. Sci. **360**(1-3), 386–414 (2006)
6. Buterin, V.: RANDAO Beacon exploitability analysis, round 2 (November 2018), https://ethresear.ch/t/randao-beacon-exploitability-analysis-round-2/1980
7. Buterin, V.: RNG exploitability analysis assuming pure RANDAO-based main chain (November 2018), https://ethresear.ch/t/rng-exploitability-analysis-assuming-pure-randao-based-main-chain/1825
8. Buterin, V.: Validator ordering and randomness in PoS (November 2018), https://vitalik.ca/files/randomness.html
9. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.: All About Maude - A High-Performance Logical Framework, Lecture Notes in Computer Science, vol. 4350. Springer-Verlag, Secaucus, NJ, USA (2007)
10. Ethereum Foundation: Announcing beneficiaries of the Ethereum Foundation grants (11 2018), https://blog.ethereum.org/2018/03/07/announcing-beneficiaries-ethereum-foundation-grants
11. Ethereum Foundation: Ethereum 2.0 spec – Casper and Sharding (11 2018), https://github.com/ethereum/eth2.0-specs/blob/master/specs/beacon-chain.md
12. Kumar, N., Sen, K., Meseguer, J., Agha, G.: A rewriting based model for probabilistic distributed object systems. In: Proc. of FMOODS '03. Lecture Notes in Computer Science, vol. 2884, pp. 32–46. Springer (2003)
13. Meseguer, J.: Rewriting as a unified model of concurrency. In: Proceedings of the Concur'90 Conference, Amsterdam, August 1990. Lecture Notes in Computer Science, vol. 458, pp. 384–400. Springer (1990)
14. Meseguer, J.: Conditional rewriting logic as a unified model of concurrency. Theor. Comput. Sci. **96**(1), 73–155 (1992). https://doi.org/http://dx.doi.org/10.1016/0304-3975(92)90182-F
15. Meseguer, J.: Membership algebra as a logical framework for equational specification. In: Parisi-Presicce, F. (ed.) Proc. WADT'97. Lecture Notes in Computer Science, vol. 1376, pp. 18–61. Springer (1998)
16. Qian, Y.: RANDAO: A DAO working as RNG of Ethereum (November 2018), https://github.com/randao/randao/
17. Sen, K., Kumar, N., Meseguer, J., Agha, G.: Probabilistic rewrite theories: Unifying models, logics and tools. Tech. Rep. UIUCDCS-R-2003-2347, University of Illinois at Urbana Champaign (May 2003)