On a decentralized trustless pseudo-random number generation algorithm

Serguei Popov

March 9, 2016

Department of Statistics, Institute of Mathematics, Statistics and Scientific Computation, University of Campinas – UNICAMP, rua Sérgio Buarque de Holanda 651, 13083–859, Campinas SP, Brazil
e-mail: `popov@ime.unicamp.br`

Abstract

We construct an algorithm that permits a large group of individuals to reach consensus on a random number, without having to rely on any third parties. The algorithm works with high probability if there are less than 50% colluding parties in the group. We describe also some modifications and generalizations of the algorithm.

Keywords: public randomness, collusion, algorithm works w.h.p.

1 Introduction and description of the basic algorithm

Suppose that there is a (large) collection of n individuals which want to reach consensus on a random number $s \in \{0,1\}^N$, but, in general, trust neither each other, nor any third parties. The outcome should have the uniform distribution, be unpredictable for everybody until revealed, and the whole procedure should be transparent/verifiable to both participants and outsiders.

Assume also that, among them, there are pn colluding parties that want to mess with the procedure (e.g., force the "random" outcome to have a specific value, or at least make it biased), where $p \in (0,1)$. We suppose that they can exchange information freely and secretly from the others, and can agree on a common strategy. Also, we make a "worst-case" assumption that, on any stage of the algorithm, the colluding parties may first wait for the others' actions, and then choose what to do accordingly to the information they currently have.

So, is it possible to do this random number generation under some reasonable assumption on the number of colluding parties? First solution that comes to mind is to ask the parties to choose their random numbers, reveal all them at the same time, and then just apply the exclusive "or" (XOR) operation to all these numbers. This, however, has a drawback that, in practice, one cannot enforce that this revelation act happens precisely at the same moment for all parties involved. The last one to reveal has, in fact, total control on the outcome, which can be exploited by a malicious player. Even if we make the parties *commit* on their numbers, the last one still has an option of never revealing it, thus effectively introducing a bias to the final result. Therefore, more complicated procedures have to be considered. One possible approach is to introduce *security deposits* (cf. e.g. [1]): each party makes one before the procedure, and does not receive it back in case it does not reveal its secret number. There is, however, a problem with this approach: if the deposits are small, then this will not prevent cheating in case the stakes are high; if the deposits are large, people may not be willing to take part in such a procedure ("what if my connection goes down between 1st and 2nd stages?")[1]. Some solutions based on public ledgers (such as the Bitcoin blockchain) were proposed, see [2]. An interesting algorithm was proposed in [4]: first, inputs from many different parties are collected, and the outcome is then computed as a deterministic function of them; this computation, however, is deliberately made so slow that no party is able to influence the result by submitting an entry at the last moment. The paper [7] mentions a procedure that apparently has some similarities with the one of the present paper, but without giving many details. We refer to the aforementioned papers for more discussion on public randomness and related topics.

In the sequel, we describe an algorithm that permits to achieve this goal with high probability. First, each party secretly chooses its number from some high-entropy random source. Then, it should commit on that number, in some usual way: for instance, publishing its hash. In practice, when the random number generation has to be performed many times in a row, the parties may commit on a last number of a long hash chain (so that the previous number is at the same time a commitment on the next one). That

[1]besides, in the latter case nodes can be blackmailed, e.g. "pay me half of your reward, or I will DDOS you between 1st and 2nd stages"; or (a bit more elegant way to state the last proposal) "pay me for my exclusive DDOS protection, because, you know, all sort of things are happening around"

is, each party then has options to reveal or withhold its number, but cannot lie about it.

Next, let us randomly divide the crowd into n/k groups of size k (for simplicity, we assume that n/k is integer). Such a division can be done in a decentralized and verifiable way that cannot be manipulated too much. For example (think about cryptocurrency accounts, such as Eth or Nxt), hash the public keys of the parties' accounts with some (sufficiently random, but public) seed[2] and order the results; then form the groups based on that order. Now, our basic algorithm is described in the following way: first, the secrets are shared within each group (so that any party that decides to withhold its number will be eliminated on this step), and then any representative of a group where this procedure did succeed (all members shared their numbers) reveals the numbers and everything is XORed together. More specifically:

1. Members of each group share their numbers between them. One way to do this would be that a party encrypts its secret number using the public keys of the other $k-1$ members of its group separately, and then broadcasts the $k-1$ outcomes.

2. Then, the group publishes a statement like "I know the secrets of all my group" signed by everybody. It is unclear whether this step is strictly necessary, but it is probably not a bad idea to do this way to assure that the consensus on which groups successfully did the secrets' sharing is reached.

3. Any group that did not publish such a statement is eliminated. In fact, any party can eliminate the group it belongs to by publishing a signed statement that the secret sharing did not succeed within that group[3].

4. Then, at least one representative of each group that was not eliminated publishes all the numbers shared by the group members.

5. All the published numbers are XORed to obtain the final outcome.

Now, this algorithm indeed achieves our goals if

(a) there is at least one group which contains only honest[4] members (so that the colluding parties cannot know their numbers beforehand);

[2]e.g., generating signature of the last block of the blockchain, or the random number obtained on the previous run of this algorithm, etc.

[3]this can be useful if a party suspects that the other members of the group are deliberately delaying the procedure

[4]by definition, a party is honest if it follows the protocol without colluding with other parties

(b) no group consist entirely of colluding parties (otherwise, such a group could introduce a bias to the final outcome by not revealing their numbers).

We denote the event from (a) by A, and the event from (b) by B. Also, let us introduce the events

$$A_j = \{\text{all parties of } j\text{th group are honest}\},$$
$$B_j = \{\text{at least one party of } j\text{th group is honest}\}$$

for $j = 1, \dots, n/k$, so $A = \bigcup_{j=1}^{n/k} A_j$, and $B = \bigcap_{j=1}^{n/k} B_j$.

Now, we need to choose k in such a way that $\mathbb{P}[A \cap B]$ is close to 1. Let us try to set $k = c \ln n$, where $c > 0$ is a parameter[5], and estimate the probabilities of the above events. Also, instead of fixing the number of colluding parties, we rather just assume that each party is malicious with probability p, independently of the others, and all the malicious parties collude. Clearly, the situation remains essentially the same, but the calculations become much easier.

First, we clearly have $\mathbb{P}[A_j] = (1-p)^k = (1-p)^{c \ln n} = n^{-c \ln(1-p)^{-1}}$, so we can write

$$\begin{aligned}
\mathbb{P}[A] &= 1 - \mathbb{P}\Big[\bigcap_{j=1}^{n/k} A_j^{\complement}\Big] \\
&= 1 - \big(1 - (1-p)^k\big)^{n/k} \qquad (1) \\
&= 1 - \big(1 - n^{-c\ln(1-p)^{-1}}\big)^{\frac{n}{c \ln n}} \\
&\simeq 1 - \exp\Big(-\frac{n^{1-c\ln(1-p)^{-1}}}{c \ln n}\Big), \qquad (2)
\end{aligned}$$

and the value of the above expression is quite close to 1 if $c \ln(1-p)^{-1} < 1$ and n is large enough.

Analogously, it holds that $\mathbb{P}[B_j] = 1 - p^k = 1 - n^{-c \ln p^{-1}}$, so

$$\begin{aligned}
\mathbb{P}[B] &= \big(\mathbb{P}[B_1]\big)^{n/k} \\
&= (1 - p^k)^{n/k} \qquad (3) \\
&= \big(1 - n^{-c \ln p^{-1}}\big)^{\frac{n}{c \ln n}} \\
&\simeq \exp\Big(-\frac{n^{-c \ln p^{-1}+1}}{c \ln n}\Big)
\end{aligned}$$

[5]to keep the things simple, we pretend in our calculations that $c \ln n$ is integer

$$\simeq 1 - \frac{n^{-c \ln p^{-1}+1}}{c \ln n} \tag{4}$$

if $c \ln p^{-1} > 1$. That is, we obtain that (at least for large n) c must satisfy the following condition:

$$\frac{1}{\ln p^{-1}} < c < \frac{1}{\ln(1-p)^{-1}}, \tag{5}$$

which is clearly possible if and only if $p < 0.5$. So, we have just proved the following result:

Proposition 1.1. *Assume that $p < 0.5$. Then*

$$\mathbb{P}[A \cap B] \gtrsim 1 - \exp\Big(- \frac{n^{1-c\ln(1-p)^{-1}}}{c \ln n} \Big) - \frac{n^{-c \ln p^{-1}+1}}{c \ln n}. \tag{6}$$

Now, since p is generally unknown, we need to find a way to make a choice for $k = k(n)$ (that is, we need to chose $c = \frac{k}{\ln n}$), that does not depend on p. One possible way to do this would be fixing a "reasonable" value of $p < 0.5$ (e.g., $p = 0.1$), and then find c that maximizes the expression in the right-hand side of (6). However, this may not be the best solution, due to the following reason: if the event $A^{\complement}$ occurs, this is much worse (from the point of view of honest parties) than the occurrence of $B^{\complement}$. Indeed, if there is a group that consists entirely of colluding parties (i.e., $B^{\complement}$ occurs), then on the last stage of the procedure they have an option of not revealing their numbers at all (after waiting all others to reveal), thus influencing the final outcome. However, the bias introduced this way is typically not so strong (there are only two options for the group, reveal or not reveal), and, perhaps more importantly, the act of not revealing their numbers is practically a confession "we are all malicious"[6]. In an ambiance where reputation (in any reasonable sense of this word) matters, such a thing could be quite harmful from the point of view of the colluding parties[7].

On the other hand, on the event $A^{\complement}$, there will be at least one "spy" in each group, i.e., the colluders will know *all* secrets already on the first stage! Of course, this opens many more possible ways for cheating: the colluding parties may eliminate groups in any possible combinations thus making quite broad adjustments to the final outcome, all this without raising a lot of suspect.

[6]while it is not unusual that a single party can go offline, it is extremely unlikely that all the members of that group went offline at once just after signing the statement that they knew all the group's secrets

[7]think e.g. about delegates in a DPoS cryptocurrency, like BTSX; they can lose their privileged positions in case of misbehaviour

p	n	k	$\alpha = \mathbb{P}[A^{\complement}]$	$\beta = \mathbb{P}[B^{\complement}]$
0.1	60	6	0.00051	1×10^{-5}
0.1	60	5	2.2×10^{-5}	0.00012
0.1	60	3	4.6×10^{-12}	0.0198
0.2	60	6	0.0478	0.00064
0.2	60	5	0.00853	0.00383
0.2	60	3	5.8×10^{-7}	0.148
0.2	120	8	0.063	3.8×10^{-5}
0.2	120	6	0.00228	0.00128
0.2	120	5	7.2×10^{-5}	0.00765
0.3	120	8	0.41	0.00098
0.3	120	6	0.081	0.0145
0.3	120	5	0.0121	0.0567

Table 1: A few numerical examples

So, an adequate way for choosing $k(n)$ is rather the following: first, we fix some $p < 0.5$ that we believe to be an upper bound on the proportion of colluding parties. Then, we decide on the acceptable values of $\alpha = \mathbb{P}[A^{\complement}]$ and $\beta = \mathbb{P}[B^{\complement}]$, for instance, $\alpha = 0.005$ and $\beta = 0.05$. If n is fixed, in general we can hope to control only one of the quantities α, β (observe that, when n is fixed and k increases, this causes α to increase and β to decrease). If one wants to control both quantities at once, one may need to increase n. See Table 1 for some numerical examples. Clearly, in practice it is better to use the exact formulas (1) and (3) for calculations.

Notice that (recall (5)) $c = \frac{1}{\ln 2} \approx 1.443$ works for any $p < 0.5$. In particular (recall (2) and (4)), the good news is that the decay of $\mathbb{P}[A^{\complement}]$ is much more rapid than the decay of $\mathbb{P}[B^{\complement}]$ (stretched exponential vs. polynomial). So, choosing $k = \lfloor 1.4 \ln n \rfloor$ for given n is probably a good rule of thumb ($\lfloor \cdot \rfloor$ stands for lower integer part, $\lfloor x \rfloor$ is the largest integer not exceeding x). Observe that this rule gives $k = 5$ for $n = 60$ and $k = 6$ for $n = 120$, compare to Table 1.

Also, it should be observed that, although α and β can be made arbitrarily small for any $p < 0.5$ (as we have just shown), the corresponding value of n can be quite large (if p is very close to 0.5).

2 Some modifications and generalizations

Consider the following situation: the overall proportion of colluding parties is small, but the nodes of the network are frequently offline, so, with high probability, no group consists entirely of active (i.e. not offline) parties. In this case, the algorithm of Section 1 will just halt, that is, will not produce any outcome. Even worse, for some values of the parameters it may happen that, with high probability, there are still some complete groups (i.e., everybody in the group is online), but each group does contain at least one malicious party (and, as discussed above, this means giving almost total control to the colluding parties)[8].

Therefore, in some situations it may be impractical to only accept the complete groups. In this section, we consider some modifications of the previous algorithm that address this issue.

The initial setup is the same: n parties are divided into n/k groups of size k, and we assume that they all have committed on their secret numbers. Then, again, they attempt to share their numbers between them, but now there is no requirement that only complete groups pass to the next round; instead, we consider the group *valid* if the number of parties that shared their secrets[9] is greater than $k/2$ (just to avoid dealing with several "conflicting" subgroups of the same group).

Then, this algorithm works fine if

(a) there is at least one group such that more than $k/2$ of its members are online and all of them are honest;

(b) no group contains more than $k/2$ malicious members.

Similarly, we denote the event from (a) by $\widetilde{A}$, and the event from (b) by $\widetilde{B}$. Also, let us introduce the events

$$\widetilde{A}_j = \{\text{all parties of } j\text{th group are honest}\\ \text{and more than } k/2 \text{ of them are online}\},$$

and

$$\widetilde{B}_j = \{\text{less than } k/2 \text{ parties of } j\text{th group are malicious}\}$$

[8] fortunately enough, one can detect such undesirable situations, because then the number of complete groups would be typically very low in comparison to n/k

[9] they have to sign and publish a statement of the sort "We are $\{i_1, \dots, i_m\}$ and all the secrets of this subgroup are known to all its members"

for $j = 1, \dots, n/k$, so, as before, $\widetilde{A} = \bigcup_{j=1}^{n/k} \widetilde{A}_j$, and $\widetilde{B} = \bigcap_{j=1}^{n/k} \widetilde{B}_j$. Again, we intend to choose $k = c \ln n$ in such a way that $\mathbb{P}[\widetilde{A} \cap \widetilde{B}]$ is close to 1. We keep the assumption that p is the probability that a party is malicious; but we assume also that a honest party is offline with probability $r > 0$ (that is, a party is malicious, honest but offline, honest and online with probabilities p, $(1-p)r$, $(1-p)(1-r)$ correspondingly).

Let us denote by

$$\Phi(k, q, s) = \sum_{\ell < s} \binom{k}{\ell} q^\ell (1-q)^{k-\ell}$$

the probability that the value of a binomial $\mathcal{B}(k, q)$ random variable is less than s. Now, let us recall the usual Chernoff's bound for the binomial distribution[10]: for any k and a with $0 < a < q < 1$, we have

$$\Phi(k, q, ak) \leq \exp\big(-kH(a, q)\big), \tag{7}$$

where

$$H(a, q) = a \ln \frac{a}{q} + (1-a) \ln \frac{1-a}{1-q} > 0.$$

Also, it is easy to see that $\Phi(k, q, ak)$ is close to 1 when $a > q$ (in that case, one may write $\Phi(k, q, ak) = 1 - \Phi(k, 1-q, (1-a)k)$ and apply the above estimates).

We have

$$\begin{aligned} \mathbb{P}[\widetilde{A}_j] &= \mathbb{P}[A_j]\, \mathbb{P}[\text{more than } k/2 \text{ are online} \mid A_j] \\ &= (1-p)^k \Phi(k, 1-r, k/2). \end{aligned} \tag{8}$$

Assume for simplicity that $r < \frac{1}{2}$. Then, the last term in the right-hand side of (8) is close to 1. So, in this case we have $\mathbb{P}[\widetilde{A}_j] \simeq \mathbb{P}[A_j]$, and therefore (2) still holds.

Next, we use (7) with $k = c \ln n$, $q = 1-p$, and $a = \frac{1}{2}$ to obtain that (note that $H(a, q)$ then equals $\frac{1}{2} \ln(4p(1-p))^{-1}$)

$$\begin{aligned} \mathbb{P}[\widetilde{B}_j] &= 1 - \Phi(k, 1-p, k/2) \\ &\geq 1 - \exp\big(-c \ln n \times \tfrac{1}{2} \ln(4p(1-p))^{-1}\big) \\ &= 1 - n^{-\frac{c}{2} \ln(4p(1-p))^{-1}}, \end{aligned}$$

[10]see e.g. Proposition 5.2 of Chapter 8 of [5], or Section 6 of Chapter I of [6]; also, the inequality in (7) is, in some sense, "almost equality", but more advanced means are needed to justify that, see [3]

so

$$\begin{aligned}\mathbb{P}[\widetilde{B}] &= \big(\mathbb{P}[\widetilde{B}_1]\big)^{n/k}\\ &= \big(1 - n^{-\frac{c}{2}\ln(4p(1-p))^{-1}}\big)^{\frac{n}{c\ln n}}\\ &\gtrsim \exp\Big(-\frac{n^{1-\frac{c}{2}\ln(4p(1-p))^{-1}}}{c\ln n}\Big)\\ &\simeq 1 - \frac{n^{-\frac{c}{2}\ln(4p(1-p))^{-1}+1}}{c\ln n} \qquad (9)\end{aligned}$$

if $\frac{c}{2}\ln(4p(1-p))^{-1} > 1$. That is, we obtain that (for large n) c must satisfy the following condition:

$$\frac{2}{\ln(4p(1-p))^{-1}} < c < \frac{1}{\ln(1-p)^{-1}}. \qquad (10)$$

The left-hand side of (10) increases when $p \in (0, \frac{1}{2})$, and the right-hand side decreases; so, the solution exists for $p < 0.2$ (if $p = 0.2$, both terms become equal). So, we have just proved the following result:

Proposition 2.1. *Assume that $p < 0.2$. Then*

$$\mathbb{P}[\widetilde{A}\cap\widetilde{B}] \gtrsim 1 - \exp\Big(-\frac{n^{1-c\ln(1-p)^{-1}}}{c\ln n}\Big) - \frac{n^{-\frac{c}{2}\ln(4p(1-p))^{-1}+1}}{c\ln n}.$$

Let us also briefly comment on the case $r > \frac{1}{2}$. The last term in the right-hand side of (8) then would also be polynomially small in n, and (7) can be used to estimate it (as we commented, the relation (7) is, in fact, "almost equality", so it gives essentially the correct order of decay). In the same way, one can arrive to a modified version of (10), with $\big(\ln(1-p)^{-1} + \frac{1}{2}\ln(4r(1-r))^{-1}\big)^{-1}$ in the right-hand side. This, in its turn, leads to a more complicated existence condition involving p and r, which we prefer not to write in an explicit way.

Next, as in Section 1, we can argue that $\widetilde{A}^{\complement}$ is much worse than $\widetilde{B}^{\complement}$; Also, all the past discussion about how to choose n and k remains valid. Observe, however, that the algorithm we just considered is less "robust" than the one of the previous section, since it can fence off at most 20% of colluding parties (vs. formerly 50%). Let us now briefly mention some further modifications/generalizations of the algorithm, that aim to increase its robustness. We do not present any further computations; we hope that the reader agrees that the corresponding asymptotic analysis (as $n \to \infty$) can be done in the same way as above.

So, first, we may consider a subgroup (where all secrets were shared) valid if there are at least γk members, where $\gamma \in (0, 1)$ (for $\gamma \in (0, \frac{1}{2})$ we need also to introduce some rules about which subgroup of a given group should win if there are several of them). One can obtain that for $\gamma \in (\frac{1}{2}, 1)$ the algorithm becomes resistant against a proportion p_γ of colluding parties, where $0.2 < p_\gamma < 0.5$.

Also, on the second stage, before XORing all the revealed numbers, we may first eliminate, say, some fixed proportion of the lower-sized valid subgroups. This gives some additional chances to get rid of all-malicious subgroups, since those must typically be of smaller size.

Another possibly useful observation is the following. Notice that it may be impractical to deal with *very* large number of parties, due e.g. to the connection/synchronization issues. However, we can take a larger crowd first, and then choose a proportion of it at random. Thus, if some party wants to mess with this, it would need to bribe really a lot of other parties.

Conclusions

1. We presented an algorithm that, with high probability, allows a large number of parties to agree on a random number in a decentralized and trustless way.

2. Our basic algorithm is described in the following way: first, each party chooses its secret number from some high-entropy source of randomness, and commits on it. Then, we form groups (of equal sizes) of parties, and the secrets are shared within each group (so that any party that decides to withhold its number will be eliminated on this step). Next, any representative of a group where this procedure did succeed (i.e., all members shared their secret numbers) reveals the numbers and everything is XORed together.

3. Under the assumption that the proportion of colluding parties does not exceed 50%, it is possible to show that the group size can be chosen in such a way that, with high probability, there is at least one group consisting entirely of honest parties, and no group consists entirely of colluders. This ensures that the algorithm works as intended.

4. In practice, there can be no "universal" rule on how to choose n and $k = k(n)$, but a good rule of thumb is choosing n to be as large as possible, and $k = \lfloor 1.4 \ln n \rfloor$.

5. We analyse also a modification of the above algorithm, where the requirement that all members of the group must share their secrets is relaxed. This may be useful when dealing with situations when honest parties are frequently offline. We also propose some further modifications and generalizations.

References

[1] `http://ethereum.stackexchange.com/questions/191/how-can-i-securely-generate-a-random-number-in-my-smart-contract`

[2] J. Bonneau, J. Clark, and S. Goldfeder (2015) On Bitcoin as a public randomness source. `http://eprint.iacr.org/2015/1015`

[3] A. Dembo, O. Zeitouni (2010) *Large Deviations Techniques and Applications.* Springer.

[4] A.K. Lenstra and B. Wesolowski (2015) A random zoo: sloth, unicorn, and trx. `http://eprint.iacr.org/2015/366.pdf`

[5] Sheldon M. Ross (2009) *A First Course in Probability.* 8th ed.

[6] A.N. Shiryaev (1996) *Probability.* Springer-Verlag, New York.

[7] E. Syta, I. Tamas, D. Visher, D.I. Wolinsky, L. Gasser, N. Gailly, B. Ford (2015) Keeping authorities "honest or bust" with decentralized witness cosigning. `http://arxiv.org/pdf/1503.08768v2.pdf`