On the Security of Election Audits with Low Entropy Randomness

Eric Rescorla

RTFM, Inc.
ekr@rtfm.com

Abstract

Secure election audits require some method of randomly selecting the units to be audited. Because physical methods such as dice rolling or lottery-style ping pong ball selection are inefficient when a large number of audit units must be selected, some authors have proposed to stretch physical methods by using them to seed randomness tables or random number generators. We analyze the security of these methods when the amount of input entropy is low under the assumption that the attacker can choose the audit units to attack. Our results indicate that under these conditions audits do not necessarily provide the detection probability implied by the standard statistics. This effect is most pronounced for randomness tables, where significantly more units must be audited in order to achieve the detection probability that would be expected if the audit units were selected by a truly random process. It is still unclear whether there are practical methods for safely using such tables for this application.

1 Introduction

Randomized audits are becoming an increasingly popular method of verifying election results. In a typical audit, preliminary results for each precinct are posted and then a fixed percentage of precincts are selected for audit. Those precincts are then manually recounted and the results are compared with the machine count. Depending on the county, discrepancies may lead to escalated auditing or simply be resolved in favor of the manual count.[1]

Unlike many statistical audit situations (e.g., quality control), election auditing is an adversarial situation and thus more care than usual must be exercised in the sample selection procedure. Consider what happens if an attacker knows that only precincts 1, 2, and 9 will be audited; he can then attack other precincts without fear of detection via the audit. Similarly, if an attacker can influence the selection of audit units, he might be able to prevent precincts that he has attacked from being audited, thus concealing evidence of the attack.

[1]Hall [8] provides a good summary of the procedure used in the California One Percent Manual Recount.

In order to prevent these attacks, it is important that the precincts which will be subject to audit be unpredictable. While there are well-known techniques for generating random numbers using dice rolling [1] and numbered ping pong balls [6], the overhead of these mechanisms is relatively high and scales linearly with the number of audit units which must be selected. For example, Calandrino et al. [3] describe plausible scenarios where hours of dice rolling might be required to perform a single audit. In order to minimize this overhead, there has been interest in methods for leveraging a small initial random value generated using one of these physical mechanisms to mechanically generate a series of "random" values which are then used to select the audit units.

As suggested by Calandrino et al., one natural approach is to use a computer-based *pseudorandom number generator* (PRNGs). PRNGs take a random *seed* value and generate an arbitrary number of pseudorandom values which can then be used for audit unit selection. Unfortunately, cryptographically secure PRNG algorithms (CSPRNGs) are generally too complicated to compute by hand and so must be executed by computer software, making their correct execution difficult for average citizens to verify (though of course someone who is moderately sophisticated can implement the algorithm themselves and verify that the output matches). Rivest [11] has proposed an algorithm which can be executed using a hand calculator. However, while observers can verify that the algorithm is being executed correctly, verifying that the algorithm is actually secure requires analysis beyond the capabilities of most laymen.

An intermediate alternative, mentioned by Cordero et al. [1], is to use a preexisting table of random numbers such as "A Million Random Digits" (AMRD) [10]. The idea here is that one uses a true RNG to select the starting place in the table and then simply reads off successive table entries. If the table is widely available (as with AMRD), then observers can obtain it themselves and have reasonable confidence that they have the correct table. It should be readily apparent that a table of this kind is simply a form of PRNG—and vice-versa—with the starting place serving as a seed.[2] However, viewed in that light, it's clear that this is a quite low entropy PRNG. For instance, AMRD has 200,000 groups of 5-digit numbers. If a starting group is randomly selected, there are less than 18 bits of entropy (i.e., less than 2^{18} distinct sequences of values). By comparison, random number generators used in cryptographic applications typically try to collect closer to 256 bits of entropy. This raises the question: Does this low level of entropy leave room for attacks?

2 The Auditing Game

In order to analyze this question, we start by formalizing the situation as an "Auditing Game" played between the attacker and the auditor.

In this game, we have a set $\mathbf{U}$ of N audit units, numbered $U_0, U_1, ...U_{N-1}$. The attacker must select any subset $\mathbf{K}$ of size k of them to attack but must do so prior to the preliminary election results being posted (depending on the attack type, the attacker might need to attack before the election, but this is not relevant for our model.) After the election results are posted, a subset $\mathbf{V}$ of size v units will be randomly selected and audited. If any unit in $\mathbf{K}$ is audited ($\mathbf{V} \cap \mathbf{K} \neq \emptyset$) then the auditor detects the attack and wins. If none of the units in $\mathbf{K}$ is audited ($\mathbf{V} \cap \mathbf{K} = \emptyset$) then the attacker wins. This reflects an auditing system in which evidence of attack is investigated somehow—as for instance would be appropriate with a DRE, which should never report any discrepancies—rather than one in which the machine count is simply replaced with a manual count. In the latter case, the security guarantees provided by an audit are much weaker, since it only acts as a correction mechanism for the audited units. It should be possible to extend this model to deal with situations where errors are dealt with by escalating audits such as those described by Stark [9], but we leave that for future work.

When the sample is drawn randomly from a uniform distribution without replacement, the statistics of this setting are well-known (see, for instance [2]). The probability that the attack will be detected is given by:

$$\Pr(\text{detection}) = 1 - \prod_{i=0}^{v-1} \frac{N-i-k}{N-i} \tag{1}$$

The expected number of matches is $\frac{kv}{N}$. If $\mathbf{V}$ is truly randomly generated, then it does not matter how the attacker selects the units to attack, any set $\mathbf{K}$ is equally likely to be in the set of audited units. However, if the attacker can exactly predict the contents of $\mathbf{V}$ then he can obviously obtain an advantage. We are interested in intermediate cases where the attacker has some information but it is incomplete.

As opposed to a true random number generator, any PRNG or table-based mechanism inherently provides some information about $\mathbf{V}$: the number of distinct $\mathbf{V}$ sets $\binom{N}{v}$ is generally far greater than the size of any plausible table or even the number of distinct outputs that a PRNG can generate, so some $\mathbf{V}$s will never occur. Equally clearly, if the seed value is small enough (e.g., only one seed is possible), then there are practical attacks. The question we focus on is how small the seed needs to be before the attacker can gain a significant advantage.

3 Forms of Bias

We initially consider the simpler to analyze case of a common table of random numbers. The table consists of T randomly generated entries, with each entry being in the range $[0..N-1]$.[3] We assume that the table is publicly known prior to the audit (this is necessary to prevent insider attack via table substitution). To select a sample of size v, a random offset o in the range $[0..T-1]$ is generated, and then entries are read off one by one (wrapping around as

[2]There is a superficial relationship here to randomness extractors (see [13] for an overview), which take a small uniform random seed value and a large amount of weakly-random data and output an unbiased random value. However, in this setting, the entire point of the random number table is to avoid the use of complicated operations like those used to produce extractors. If such functions are tolerable, as discussed in Section 6, it is simpler to use the random value to directly seed the PRNG.

[3]As a practical matter, one would probably start with an existing table such as AMRD. Such a table requires some adjustment since the entries will not fall into the exact range $[0..N-1]$. However, methods are readily available [7] for making these adjustments.

necessary) until v unique entries have been collected. These then become the sample to audit. Thus, any audit sample is uniquely defined by the pair (v, o). For simplicity, we assume that the attacker knows v in advance. In jurisdictions where a fixed size audit is performed, this is always true. In jurisdictions where the size of the audit depends on the margin of victory, the attacker will likely have some idea of v from polls. The only value the attacker does not know is o, which, as we have said, is randomly generated.

Because the attacker has the table available to him in advance, he has an opportunity to analyze it for aggregate properties which he can then exploit. In the following section we describe two such properties.

3.1 Natural Variance

While in a randomly generated table the expected number of instances of each value is $\frac{T}{N}$, each individual value does not appear in the table with equal frequency. Rather, they follow the multinomial distribution, with some units appearing more often than $\frac{T}{N}$ and some appearing less often[4] This variance allows the attacker who has access to the table to gain an advantage: he simply selects the audit units which are represented least frequently in the table. In order to estimate the size of this effect, we need to ask what fraction of the *entries* in the table will be "bad" (correspond to the units under attack) if the attacker follows this strategy. In other words, what is the total number of entries in the table corresponding to the k least frequent units?

The distribution of the number of instances of any given unit follows the binomial distribution with count T and probability $\frac{1}{N}$. We denote the probability density function of that binomial distribution as $\varphi(n)$ and the cumulative distribution function as $\text{cdf}(n)$, with both functions defined on the domain $[0..T-1]$. Thus, the expected number of units that appear exactly n times is $N\varphi(n)$.

In order to estimate the number of appearances of the kth least frequent unit, we simply need to compute the n value that represents the bottom k/T fraction of the distribution. Thus, the number of appearances of that value n_k is given by:

$$n_k = \min\left\{n : \text{cdf}(n) \geq \frac{k}{N}\right\} \tag{2}$$

Similarly, we can compute the total number of appearances of the least frequent k values put together (and hence the total number of bad entries) by computing:

$$T_{bad} = N \sum_{n=0}^{n_k} n\varphi(n) \tag{3}$$

Conversely, there are $T - T_{bad}$ "safe" entries in the table, which correspond to units which have not been tampered with and therefore can be audited without fear of discovery. If we were to simply to select one entry from the table at random and audit the corresponding unit, we would thus have an T_{bad}/T chance of detecting the attack.

Because we are sampling without replacement, we must remember that each "good" sample removes some of the safe space. On average, the remaining units each appear in the table with frequency $F = \frac{T - T_{bad}}{N-k}$. Thus, each audit removes approximately fraction F of the space. Putting this all together, the probability of detection becomes approximately:

$$\Pr(\text{detection}) = 1 - \prod_{i=0}^{v-1} \frac{T - iF - T_{bad}}{T - iF} \tag{4}$$

Note the similarity in structure of Equation (4) to Equation (1). The only difference is that we are now working with table entries rather than with audit units. If we assume that all units occur equally frequently, then this equation just becomes Equation (1) with the numerator and denominator multiplied by the unit frequency of T/N.

Unfortunately, Equations (2) and (4) provide an (often significant) overestimate of the probability of detection because of quantization effects: $\text{cdf}(n)$ is only meaningful at whole integer intervals, and thus typically $\text{cdf}(n_k - 1) < \frac{k}{N} < \text{cdf}(n_k)$. So, for instance we might find that if we pick out units which occur 50 or fewer times, we get the bottom 2% of units, but if we pick out units which occur 51 or fewer times we get the bottom 4% of units. If we're looking for the bottom 3%, then, we need to select only some of the units that occur 51 times and Equation (3) gives us a significant error. We can get a better estimate by scaling the last term in the sum to match the "remainder" of the probability mass after the terms through $n_k - 1$ have been added, as shown in Equation (5)[5]

$$T_{bad} = \left(N \sum_{n=0}^{n_k - 1} n\varphi(n)\right) + n_k\left(\frac{k}{N} - \text{cdf}(n_k - 1)\right) \tag{5}$$

[4]The Wikipedia articles on the multinomial [15] and binomial [14] distributions provide a good introduction here.

[5]This equation provides results that match simulations under random sampling, but not always under samples with contiguous ranges. We are still investigating this point, but the clustering effect discussed in the next section is suspected.

0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1
6	7	8	9	2	3	4	5	6	7
8	9	2	3	4	5	6	7	8	9
2	3	4	5	6	7	8	9	2	3
4	5	6	7	8	9	2	3	4	5
6	7	8	9	2	3	4	5	6	7
8	9	2	3	4	5	6	7	8	9
2	3	4	5	6	7	8	9	2	3
4	5	6	7	8	9	2	3	4	5

Figure 1: A table constructed to minimize detection

0	2	3	4	5	1	6	7	8	9
0	2	3	4	5	1	6	7	8	9
0	2	3	4	5	1	6	7	8	9
0	2	3	4	5	1	6	7	8	9
0	2	3	4	5	1	6	7	8	9
0	2	3	4	5	1	6	7	8	9
0	2	3	4	5	1	6	7	8	9
0	2	3	4	5	1	6	7	8	9
0	2	3	4	5	1	6	7	8	9
0	2	3	4	5	1	6	7	8	9

Figure 2: A table constructed to maximize detection

Qualitatively, we can state the following:

- As the expected number of counts for each unit increases, the distribution of the number of counts for each unit tightens. Thus, larger tables offer the attacker less advantage.
- As the number of units increases, then the expected number of counts per unit for a fixed size table decreases. Thus, the attacker can obtain a higher advantage with large N (e.g., more precincts).
- Higher k values decrease the attacker's advantage because he must select increasingly probable units to attack.

3.2 Clustering

The above analysis only provides a lower bound on the attacker's edge. The second form of bias that the attacker might exploit is clustering. Just as the table entries do not appear with equal probability, they do not have uniform density throughout the table. If the audit sample is selected via consecutive ranges of entries, which is by far the simplest method, this can be exploited to depress the probability of detection by overlapping the samples that capture the regions to be attacked.

As an intuition pump, consider what happens if the attacker is allowed to construct his own table of entries with the restriction that all entries must appear with equal probability. For $T = 100$, $N = 10$, $k = 2$, $v = 5$, the table shown in Figure 1 minimizes the chance that units 0 or 1 will be selected. This construction places all values of 0 and 1 together in the table, with the result that only offsets which directly land on 0,1 or which are less than v values before the contiguous block produce a sample containing either 0 or 1. These offsets are shaded, with offsets which would produce samples containing both 0 and 1 shaded darker. Note that as any hit causes the attacker to lose the game, having both units discovered doesn't make the situation worse. The attacker thus has a 75% chance of winning. (The chance of winning for randomly chosen attack units is around 22%.) By contrast, it is easy to construct a table in which the probability of discovery is unity, simply by arranging that either a 0 or 1 appears every 5 entries, as shown in Figure 2.

Obviously, in our scenario, the attacker does not get to control the table, and the tables are much larger, so the magnitude of this attack is lessened. However, randomly generated tables contain naturally occurring clusters which can be exploited to some advantage, as we explore in the next section.

4 Simulations

While we don't yet have an analytic model for the clustering effect we can start to get a handle on the problem via simulation. The general approach is to randomly generate a table of a given size with a good PRNG (OpenSSL `RAND_bytes(3)`). We then randomly select an attack set $\mathbf{K}$ and compute the fraction of offsets in the table which would sample at least one member out of the attack set for a given audit sample size v. The result is equal to the probability of detection of attack $\mathbf{K}$ with that size audit. We ran simulations with both random attack sets and ones where the audit units were chosen to minimize detection and determined the mean detection rate (which should be equal to the expected value) as well as the chance of detection for the attack set with the lowest detection probability.

If we were to ignore the clustering effect, for instance by doing non-sequential sampling, then the attacker's best strategy would be to select the k least probable units. We don't currently know how to efficiently compute an optimal attack set in the face of clustering, but we can start by randomly sampling out of the lower tail of the audit unit frequency distribution. As a heuristic, we randomly generate k-sized subsets out of the least frequent $2k$ audit units. This lets us explore the space and select the least probable attack set out of those we generate.[6]

Each sampling procedure was repeated 10,000 times, using a separate randomly generated table for each combination of N and T. In addition, to avoid the impact of "unlucky" random tables, to generate the figures below we repeated the entire procedure for multiple randomly generated tables. Where possible, we used 25 trials, but for Table 2, Figures 4, 5, 9, and the permuted curve in Figure 7, we only ran 5 trials due to time constraints. In addition, we ran some experiments with the values in AMRD with qualitatively similar results. The AMRD [10] table seems to have slightly fewer outlying values, but that's most likely due to natural variation, not due to any defect in the tables.

4.1 Parameter Selection

As mentioned above, the usefulness of this attack depends on both the size of the random table and the number of audit units (e.g, precincts). Two natural anchor points for the random table are 200,000 and 65,000 entries, with the first value corresponding to the number of entries in AMRD and the second approximating the period of a PRNG with a 16-bit state. [Note that this is sometimes but not always the same as a PRNG with a 16-bit seed. We take up the question of PRNGs with large states but small seeds in section 6.] We also examine the case of 1,000,000 entries, which is about the largest table that can plausibly fit into a single book.

The number of precincts varies widely between different counties and states. Somewhat arbitrarily, we chose 5000 (approximately the number of precincts in Los Angeles County [4883]), 1000 (approximately the number of precincts in Santa Clara County [929]), and 100 (approximately the number of precincts in Yolo County [149]).

Note that with precinct-based audits, the attacker's optimal strategy is generally to concentrate his attacks on as few of precincts as possible, because this minimizes his chance of detection. Thus, the total amount the attacker can shift the election is bounded by the maximum amount he can shift any individual precinct without the anomaly being so great it will be detected by other means (e.g., 100% turnout for one candidate in a district that polls in favor of his opponent) multiplied by the number of precincts he attacks. It's hard to estimate this quantity, since in principle every vote for the nominal winner could actually be a vote for the nominal loser. It's common to talk about *within precinct miscount* (WPM)[12], and assume that no more than 20% of votes in a precinct can be shifted without detection. Without taking a position on the appropriate error bounds, we consider attacks on 1% and 5% of the units, which could represent shifts up up to 5% of votes, though in practice they are almost certainly lower, since it is very unusual for the nominal loser to get no votes, so even if the attacker could completely control the results he would be counting votes for the nominal winner "accurately". The rest of the paper is discussed purely in terms of the number of attacked units, with the understanding that the total vote shift is controlled by the reader's assumptions about the error bounds.

4.2 Results

Figure 3 shows the results of a typical simulation, for a 200,000 entry table with 1000 precincts, 10 of which (1%) are being attacked. The dashed line shows the mean probability of detection for a randomly selected set of precincts under attack and closely follows Equation (1). The solid line shows the minimum observed probability of detection for an attack set chosen from the least frequent precincts, as described in the previous section. The slight shakiness in this line is due to sampling error: We are exploring the lower tail of the frequency distribution and so successive simulations may get more or less lucky. This line represents an upper bound on the detection rate experienced by the attacker, because there may be even more optimal attack sets we have not explored. The dotted lines indicate 80%, 90%, 95%, and 99% detection probabilities. Note that each line represents the mean of trials with 25 separate tables. We do not show error bars because on the scale shown they track so closely to the lines that they obscure the figure.

There are two ways of looking at these results: from the perspective of the attacker and from the perspective of the auditor. From the perspective of the attacker, who cares about how much he can re-

[6]There's no dependency on this search algorithm being random; it's just that we don't have a better algorithm so this seemed like a reasonable way to explore. Finding a superior search algorithm is an open question.

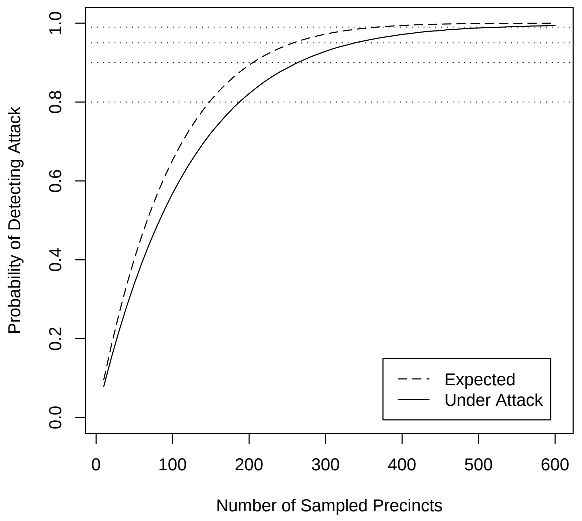

Figure 3: Probability of detection: 200,000 entries, 1000 precincts, 10 attacked precincts

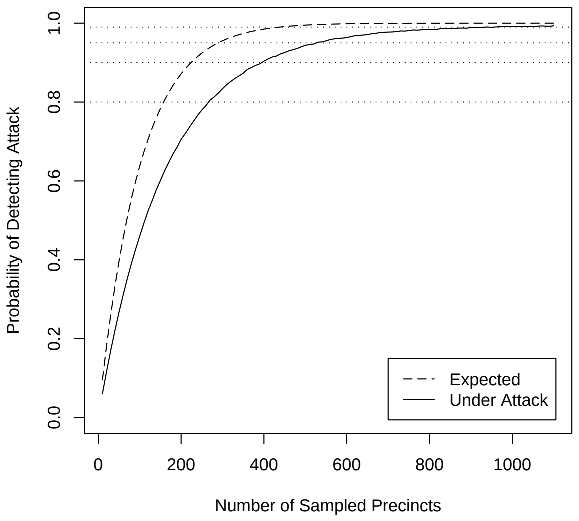

Figure 4: Probability of detection: 200,000 entries, 5000 precincts, 50 attacked precincts

duce the probability of detection, the effect is relatively modest. For an audit intended to achieve a 99% detection probability, the attacker can lower his probability of detection to approximately 96%, a factor of 4 improvement in his odds, but still a very high chance of being detected. The situation is a little better for lower intended detection probabilities, but the attacker never gains more than about a 10% absolute advantage. Whether this is significant depends on one's model of attacker behavior: it seems unlikely that the advantage is large enough to make an attacker choose to mount an attack they otherwise would not except in very marginal cases. However, if an attacker was planning to mount an attack anyway, it clearly would be to their benefit to select the audit units to attack carefully.

From the perspective of the auditor, however, the situation looks very different.[7] An auditor who wants to achieve a given detection probability must work significantly harder against an attacker who can analyze his random number generation procedure than against one who cannot. Table 1 shows the number of units which must be audited at four natural audit levels, rounded up to the nearest 10, with the analytically projected values from Equation (1) in parentheses. This effect becomes more pronounced as we approach higher intended detection probabilities and the curves start to level off; at the 99% level, the we must audit almost 50% more precincts to achieve the same detection probability.

[7]I am grateful to Hovav Shacham for suggesting this way of looking at the data.

Detection Probability	Units to Audit (random)	Units to Audit (targeted)
80%	150 (148)	190 (148)
90%	210 (205)	270 (205)
95%	260 (258)	340 (258)
<99%	370 (368)	540 (358)

Table 1: Required audit levels: 200,000 entries, 1000 precincts, 10 attacked precincts

Figure 4 shows another example, with the same table size and attack level parameters but 5,000 audit units. Here the effect is even more pronounced, with the size of the required audit set under attack being over twice as large as that which would be expected from Equation (1).

Finally, Figures 5 and 6 show two parameter sets where this attack doesn't work well. In Figure 5, we see a large table, which greatly reduces the impact of both effects. In Figure 6, despite the small table size (65,000), the small number of precincts (100) and the high attack rate (5%) result in a small advantage for the attacker. Note, however, how high the audit rate is in this case: in order to get 99% detection probability you need to audit 59 units out

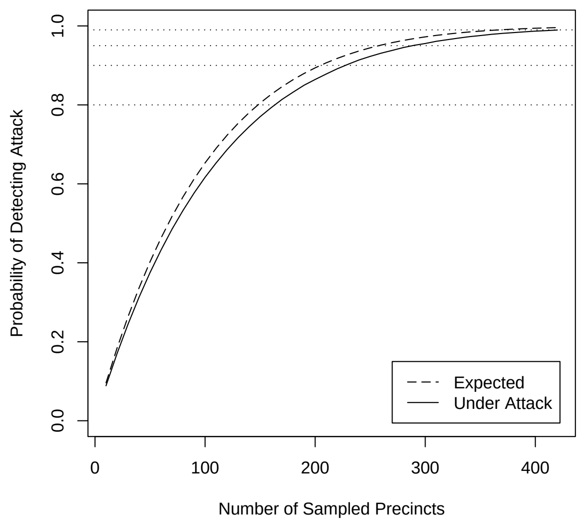

Figure 5: Probability of detection: 1,000,000 entries, 1000 precincts, 10 attacked precincts

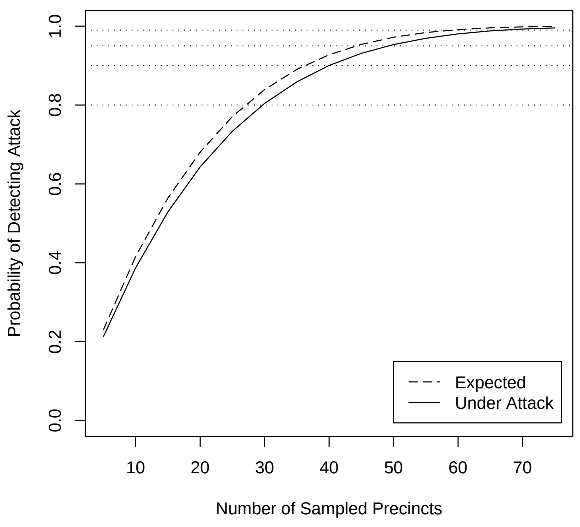

Figure 6: Probability of detection: 65,000 entries, 100 precincts, 5 attacked precincts

of 100. This is a result of the small number of audit units and reflects an unfortunate tradeoff: Dividing the election into a high numbers of audit units yields better statistical power with less total auditing[8] but makes precomputation attacks on the random number generation more powerful.

Table 2 provides a summary of the results for a variety of parameter values. The "Detected.01" and "Detected.05" columns indicate the fraction of attacks that would be detected when attacking 1% and 5% of the units respectively. Note that while the intended detection probabilities are the same, the number of units that must be audited ("Sampled.01" and "Sampled.05") is dependent on the *expected* attack level.

The results in Table 2 confirm the qualitative assertions from Section 3.1. Larger table sizes provide more of an advantage for the auditor; more audit units provide an advantage for the attacker; the more units the attacker has to attack the less advantage he can obtain. The worst case scenario, then is a table size of 65000 in a very large county, where the attacker can reduce his probability of detection to 76% under an audit designed to have 99% probability of detection if he need only attack 1% of precincts. By contrast, when there are only 100 audit units, the attacker gains a very marginal advantage at nearly every audit size.

One final question to address is the relative magnitude of the variance and clustering effects. It is difficult to provide a precise quantitative estimate but they are both significant. As discussed earlier, just selecting the least frequent k units does not produce as much of an advantage as searching through the least frequent $2k$. We can also get a qualitative impression by comparing the attacker's advantage on tables which are randomly generated versus those which are generated by randomly permuting equal numbers of each unit as shown in Figure 7. The attacker still gains an advantage with the permuted table, but it lies in between the random table and the expected probability of detection. For this set of parameters, the probability of detection with a permuted table at the nominal 99% level is around 97.6% (as compared to around 96% with the random table); the auditor needs to audit around 470 units as opposed the normal 368 units and 540 with a random table.

5 Sparse Tables

So far we have considered only dense tables: those in which the auditor can select any entry within the table as his starting point. However it is also possible to have "sparse" tables, in which only some offsets into the table are possible. For instance,

[8] I owe this formulation of this fact to Arlene Ash.

Table.Size	Num.Units	Confidence	Sampled.01	Detected.01	Sampled.05	Detected.05
1000000	5000	0.800	158	0.729	32	0.758
1000000	5000	0.900	224	0.843	45	0.863
1000000	5000	0.950	290	0.908	59	0.926
1000000	5000	0.990	438	0.971	89	0.979
1000000	1000	0.800	148	0.765	31	0.778
1000000	1000	0.900	205	0.872	44	0.882
1000000	1000	0.950	258	0.929	57	0.938
1000000	1000	0.990	368	0.981	86	0.985
1000000	100	0.800	80	0.790	27	0.791
1000000	100	0.900	90	0.892	37	0.899
1000000	100	0.950	95	0.945	45	0.948
1000000	100	0.990	99	0.988	59	0.988
200000	5000	0.800	158	0.620	32	0.687
200000	5000	0.900	224	0.740	45	0.802
200000	5000	0.950	290	0.822	59	0.880
200000	5000	0.990	438	0.922	89	0.956
200000	1000	0.800	148	0.714	31	0.744
200000	1000	0.900	205	0.831	44	0.854
200000	1000	0.950	258	0.896	57	0.916
200000	1000	0.990	368	0.961	86	0.975
200000	100	0.800	80	0.784	27	0.783
200000	100	0.900	90	0.884	37	0.893
200000	100	0.950	95	0.939	45	0.944
200000	100	0.990	99	0.986	59	0.985
65000	5000	0.800	158	0.443	32	0.568
65000	5000	0.900	224	0.559	45	0.689
65000	5000	0.950	290	0.646	59	0.781
65000	5000	0.990	438	0.773	89	0.892
65000	1000	0.800	148	0.641	31	0.698
65000	1000	0.900	205	0.762	44	0.811
65000	1000	0.950	258	0.836	57	0.885
65000	1000	0.990	368	0.925	86	0.956
65000	100	0.800	80	0.765	27	0.768
65000	100	0.900	90	0.871	37	0.880
65000	100	0.950	95	0.928	45	0.934
65000	100	0.990	99	0.978	59	0.980

Table 2: Summary of Attacker Advantage: Confidence indicates the desired detection probability; Sampled.01 and Sampled.05 indicate the number of units sampled assuming a 1% and 5% attack fractions; Detected.01 and Detected.05 indicate the probability of detection at those attack fractions.

AMRD has 1000 entries per page (for a total of 200 pages). If, instead of selecting a random entry, we just select a random page and start at the top of the page, then there are only 200 possible offsets, $0, 1000, 2000, ...200,000$.

More formally, as before we have a table containing T randomly generated entries, but instead of choosing any offset in the range $[0..T-1]$, the auditor only chooses randomly from some smaller set of offsets. Intuitively, this weakens the security of the system. Consider the limiting case where there are only two possible offsets, 0 and $100,000$; the attacker gets quite a large amount of information about which units will be audited. Indeed, if $v < \frac{1}{2}N$, there will always be some units which are not audited at all and so can be attacked safely.

Even in less extreme cases there is a substantial loss of security. Figure 8 shows the same parameters as Figure 3, but with only 200 offsets, simulating the use of AMRD with page level addressing. The detection rate under attack with normal addressing is shown for reference. While the *expected* behavior of this addressing mode is the same, the attacker gains significantly more advantage, with only about a 92% chance of detection at the nominal 99% level and 640 units which must be sampled to achieve a 99% detection probability.

6 Cryptographically Strong PRNGs

We now take up the question of PRNGs. As mentioned in Section 1, these can be thought of as a large table, with the seed selecting an offset into the table. The way that PRNGs are typically implemented is that there is some internal state value s which can take on any value in the range $[0..S-1]$; an output function $f()$; and a transition function $\delta()$; as shown below:

To extract one value from a PRNG in state s, we first compute $f(s)$ to get the output and then set $s = \delta(s)$ to determine the next state. It should be apparent that if the state value has maximum size S than no more than S values may be extracted before

[9] Actually performing this rearrangement is deliberately impractical with any strong PRNG, but we're just using this notation for analytical simplicity.

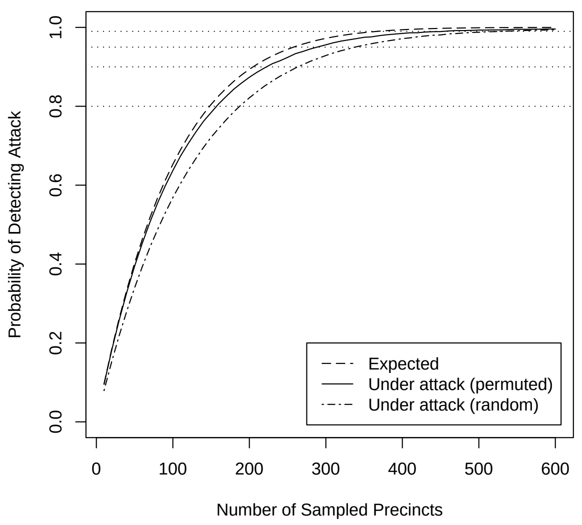

Figure 7: Probability of detection: 200,000 entries, 1000 precincts, 10 attacked precincts, permuted vs. random

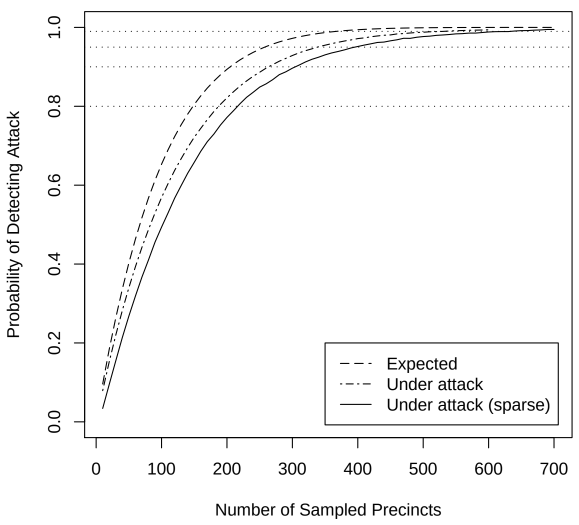

Figure 8: Probability of detection: 200,000 entries, 1000 precincts, 10 attacked precincts, 200 offsets

the output pattern starts to repeat. Ideally, this "cycle length" would be approximately S, but in some inferior PRNGs, it is actually far smaller. For the purposes of this discussion, we ignore such PRNGs. Without loss of generality, then, we can simply reorder the states so that they are numbered $[0..S-1]$ with $\delta(s) = (s+1) \bmod S$.[9]

Of course, before a PRNG can be used, we must first select the initial state (the "seed")—if the same initial state is always used then the output of the PRNG is perfectly predictable. To use a PRNG in an audit system, one would generate the seed in some publicly verifiable way and then use the PRNG to generate the audit units. The case we are interested in is that where the size of the seed is much smaller than S. Internal states are typically on the order of several hundred bits and generating that much randomness by dice rolling, coin flipping, etc. seems impractical. For instance, to generate 160 bits of entropy would require 54 rolls of 8 sided dice. Obviously, this means that some s-values are not accessible as initial states.

What does this mean for the security of an auditing system? First, if S is much larger than the seed space, the table is likely to be very sparse. In principle, it is possible to construct a PRNG so that the seed space would map to a small, adjacent, set of internal states, so that if you started with seed i, and hence state s_i and then extracted some small number of values, you would end up in internal state s_j corresponding to seed j. Effectively, this would be a PRNG with a much smaller state space—about the same size as the seed space. In that case, the analysis of the previous sections would hold and the attacks we have already discussed would be equally effective. However, the design of CSPRNGs generally ensures that even small input ranges are mapped across the entire internal state space, and so the probability of overlaps between the unit sequences induced by various seeds will be negligible. Increasing S beyond the point where $\frac{S}{N} >> v$ has negligible effect on security; it simply results in a large number of states which can never be reached in any audit.

It's easiest, then, to model a PRNG as a set of independent tables containing $[0..N-1]^*$ and indexed by seed. Since each unit can only be sampled once, we can simply regard each of these tables as a permutation of $[0..N-1]$, with a (v,o) audit selecting the first v units in table o. Because this is effectively a much larger table (N times the seed size), even with a small seed (e.g., 16 bits), such a PRNG provides a fairly good level of security. Figure 9 shows the probability of detection for seeds from 1–16 bits with an audit sized at the nominal 99% level. Note that we are using a slightly different search algorithm here: instead of looking for the least common

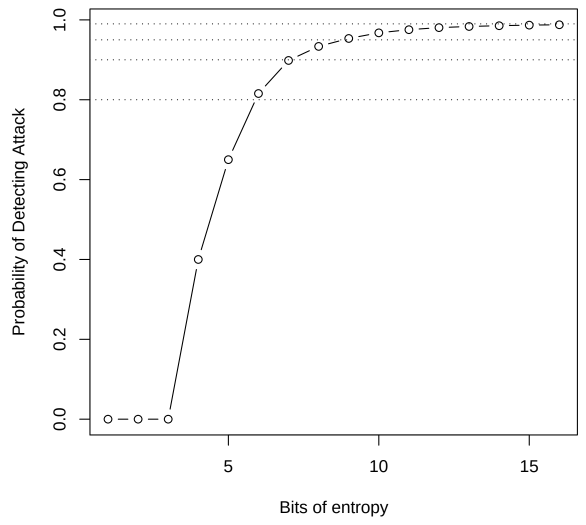

Figure 9: Probability of detection for PRNGs: 1000 precincts, 10 attacked precincts

units (as all are equally frequent), we instead look for the ones that appear least frequently in the first v entries for each seed (recall that the sequences for adjacent seeds do not overlap.) This does not have a significant impact on large, overlapping tables like the ones we considered in Section 4.2, but does make a difference here.

As shown in Figure 9, for PRNGs with very small seed the attacker is actually able to find units which *never* appear in the first v entries for each seed, thus obtaining a detection probability of 0. As expected, as the seed space increases, we approach the expected security level. With a 16-bit seed, we see a detection probability of 98.9, which is only a very marginal advantage for the attacker. By contrast, the 65,000-entry tables shown in Table 2, which models a PRNG with a small state space as well as a small seed space, provide the attacker with a substantial advantage. This reinforces the importance of using a PRNG with a large state space even if the seed space is small.

7 Potential Approaches

It is clearly very desirable to have some mechanism for stretching a small amount of verifiable public randomness into an arbitrary number of audit selections. However, as we have shown, some natural approaches provide a lower probability of detection than would be expected from the conventional statistical models of auditing, which assume perfect randomness. In this section we consider some variants of these mechanism which potentially offer higher detection levels.

7.1 Randomness Tables

While randomness tables are potentially safe as a mechanism for generating audit units, AMRD provides the attacker with a significant advantage in a number of plausible settings. Any table provides the attacker with some advantage, but a large enough table can potentially reduce the advantage to the acceptable level. Unfortunately, the table has to be very large; as Table 2 shows, even a table of size 10^6 provides the attacker a significant advantage in large jurisdictions. Equation (4) suggests that a table of 10^7 may be sufficient (the attacker's advantage at the 99% level with 5000 precincts and 1% of units attacked is $< .5\%$) however, this ignores the clustering effects and in any case a table of size 10^7 would be 10,000 pages using the AMRD formatting (2,000 pages if we were to address individual digits as described below), which seems on the edge of prohibitively large.

The situation can be substantially improved if we substitute a randomly permuted table containing equal frequencies of each entry. [For instance, fill entry i of the table with $i \bmod N$ and then randomly permute. Knuth [4] provides a suitable permutation algorithm.] This eliminates the frequency variance effects leaving us only with clustering effects (see Figure 7 for an example). While we do not yet have an analytic model for the clustering effects, initial simulations suggest that a table of size 10^6 is sufficiently large to minimize them, though 10^7 is probably safer.

While it is possible to create a truly random table using updated versions of the methods used to produce AMRD, this is likely to be a relatively expensive proposition. An alternative would be to use a CSPRNG to produce the digits but then print them in a book form factor. The major difficulty here is to demonstrate that the paper version accurately represents the output of the CSPRNG. There are two major approaches to this problem. The first is procedural: distribute the PDF file corresponding to the book and allow others to verify that the CSPRNG stream matches the PDF. Outside auditors could be allowed to verify that the correct PDF was used to generate the book. The second defense is technical: random auditing could be used to detect systematic errors.

The second, more minor, issue is to verify that the seeds for the CSPRNG were randomly generated. However, since this only need be done once, it is efficient to generate a large seed (> 160 bits) in a public ceremony using techniques like those described by Cordero et al. [1].

Another alternative is to try to further stretch an existing table such as AMRD. There are a number of natural strategies, suggested to me variously by Nadia Heninger, Mark Lindeman, Dennis Paull, David Wagner, and an anonymous reviewer.

Addressing individual digits rather than columns. While digits in AMRD are organized into groups of five, one more dice roll would suffice to define an offset into columns. The improvement this provides is modest: If we stick with five digits at a time it makes AMRD a 1,000,000-entry table, which, as noted above, still gives the attacker a substantial edge. In addition, it seems like it would be easy to get out of alignment, since you're regularly reading the end of one group and the beginning of another, this is an opportunity for mistakes. It would be even more confusing if you were reading less than five digits, since you then are constantly shifting the offset into the groups.

Add a random offset. Another possibility is to generate a random value which is then added to every entry in the table ($\bmod N$) before it is used to look up a precinct. One would probably need at minimum 8 bits of additional entropy (4 bits if used with individual digit addressing) in order to provide a sufficient security improvement under our model. In the limit, if the offset is chosen in the range $[0..N-1]$, then it effectively rotates the unit numbers and the result nullifies the natural variance effect. There is no point in an offset larger than this since any offset $r \geq N$ has the same effect as $r \bmod N$. This procedure also seems somewhat unwieldy, since it requires arithmetic for each entry. However with some care it might be practical.

Pick Two Starting Points An anonymous reviewer suggests using two separate starting points (potentially with two tables) and then add each value to get the unit to audit. In the best case, this seems like it squares the number of possible sequences.

Random intervals. Finally, one might imagine not selecting consecutive entries but rather generating a random value m and then choosing every mth entry. In the best case, it seems like this would simulate a table of size mT. However it seems like this would be very error-prone with values of m larger than 10–20, as it would require counting forward quite carefully over multiple pages.

In all of these cases, one might naively think that an additional input entropy of b bits would provide an effective table size of 2^b. However, because the same table entries are being reused, albeit salted with the additional entropy, further analysis is needed to determine whether this is in fact the case.

7.2 PRNGs

A PRNG provides the potential for a less unwieldy solution. While a complete prescription for selecting a PRNG for auditing applications is outside the scope of this paper, our analysis points to some minimal guidelines: such a PRNG should have a large internal state and should be designed to accept small seeds but distribute them to widely divergent internal states, thus minimizing the probability of overlap. CSPRNGs such as those specified by NIST SP 800-90 [5] are an appropriate choice for this task. Further, they should be seeded with at least 16 bits of entropy, however, 32 bits would provide a significantly greater margin of safety.[10] Other PRNGs may also be suitable, however further study would be required. In particular, SSR [11] seems like an interesting possibility, but we have not yet applied the techniques described in this paper to these alternatives.

8 Conclusions

Secure election audits rely on random selection of the units to be audited. We find that two natural methods, randomness tables and pseudorandom number generators may be susceptible to pre-analysis by attackers who can select audit units to attack which are unlikely to be audited. In such cases, randomized audits may not deliver their intended detection probability and significantly more units must be audited in order to attain the desired detection probability. It is still unclear whether there are practical methods for using random number tables for this purpose.

[10]Note that the NIST CSPRNGs are specified as requiring a very large amount of seed material. This is necessary for cryptographic applications in order to prevent attacks where the attacker exhaustively searches the key space. However, for this application we merely need the attacker to have very little information *in advance* about which units will be chosen and so a much smaller seed suffices. Thanks to an anonymous reviewer for suggesting this point.

While it is clear that with enough entropy PRNGs can be used safely, this analysis provides some lower bounds on the level of entropy that must be used.

Acknowledgements

Thanks to Joseph Lorenzo Hall for inspiring me to consider this problem and for his comments on a previous version of this paper. This work was shaped by productive discussions with Kevin Dick, Cullen Jennings, Hovav Shacham, and Terence Spies. Thanks to Nadia Heninger, Mark Lindeman, Neal McBurnett, Dennis Paull, David Wagner, and Dan Wallach for their useful comments on previous versions. Thanks also to the anonymous EVT reviewers for their helpful comments and to Steve Checkoway for comments and a close read.

This version of this paper is revision 113, and is dated June 27, 2009. A previous revision, dated June 23, was briefly available on the EVT site.

References

[1] Arel Cordero and David Wagner and David Dill. The Role of Dice in Election Audits—Extended Abstract. IAVoSS Workshop on Trustworthy Elections 2006 (WOTE 2006), June 2006. `http://www.cs.berkeley.edu/~daw/papers/dice-wote06.pdf`.

[2] J. A. Aslam, R. A. Popa, and R. L. Rivest. On estimating the size and confidence of a statistical audit. In *Proc. 2007 USENIX/ACCURATE Electronic Voting Technology Workshop (EVT '07)*, 2007.

[3] J. A. Calandrino, J. A. Halderman, and E. W. Felten. In Defense of Pseudorandom Sample Selection. In *Proceedings of the 2008 Electronic Voting Technology Workshop*, 2008. `http://www.usenix.org/events/evt08/tech/full_papers/calandrino/calandrino.pdf`.

[4] Donald E. Knuth. *The Art of Computer Programming: Seminumerical Algorithms*, volume 2. Addison-Wesley, 3 edition, 1998.

[5] Elaine Barker and John Kelsey. Recommendation for Random Number Generation Using Deterministic Random Bit Generators (Revised). NIST Special Publication 800-90, March 2007. `http://csrc.nist.gov/publications/nistpubs/800-90/SP800-90revised_March2007.pdf`.

[6] Joseph Lorenzo Hall. Research Memorandum: On Improving the Uniformity of Randomness with Alameda County's Random Selection Process, March 2007.

[7] Joseph Lorenzo Hall. Dice Binning Calculator for Post-Election Audits, March 2008. `http://www.josephhall.org/dicebins.php`.

[8] Joseph Lorenzo Hall. *Policy Mechanisms for Increasing Transparency in Electronic Voting.* PhD thesis, University of California, Berkeley, 2008. `http://josephhall.org/papers/jhall-phd.pdf`.

[9] Philip B. Stark. Conservative statistical post-election audits. *The Annals of Applied Statistics*, 2:550–581, 2008. `arxiv.org/abs/0807.4005)`.

[10] RAND Corporation. *A Million Random Digits with 100,000 Normal Deviates.* American Book Publishers, 2002.

[11] Ronald L. Rivest. Sum of Square Roots (SSR) Pseudorandom Sampling Method for Election Audits. `http://people.csail.mit.edu/rivest/Rivest-ASumOfSquareRootsSSRPseudorandomSamplingMethodForElectionAudits.pdf`, April 2008.

[12] H. Stanislevic. Random auditing of e-voting systems: How much is enough?, 2006. `http://www.votetrustusa.org/pdfs/VTTF/EVEPAuditing.pdf`.

[13] S. Vadhan. Randomness extractors & their cryptographic applications. Invited Tutorial TCC 2008, 2008. `http://people.seas.harvard.edu/~salil/papers/extractors-tcc08.pps`.

[14] Wikipedia. Binomial Distributions. `http://en.wikipedia.org/wiki/Binomial_distribution`.

[15] Wikipedia. Multinomial Distributions. `http://en.wikipedia.org/wiki/Multinomial_distribution`.