**Draft of June 7, 2019**

**Note by Ronald L. Rivest**

**For inclusion in the proceedings of the “Post-Election Auditing” summit**

**Held at MIT December 7-8, 2018.**

**Presentation title:**

“Two new election audit tools: sampling with *k*-cut, and Bayesian audits”

**Note title:**

Sampling with *k-*cut, and Bayesian audits

Ronald L. Rivest

MIT

# Introduction

We present two new tools for the election auditor’s toolbox that may provide increased efficiency, or additional flexibility in complicated situations.

Post-election statistical tabulation audits proceed by sampling cast paper ballots at random, and then figuring out what the sampled ballots tell you about the correctness of the reported election outcomes.

We propose a new procedure, *sampling with k-cuts,* for drawing random samples of cast paper ballots for such statistical post-election audits. This procedure eliminates the need to count down to a specified pseudo-random position in a stack of ballots, by performing instead a sequence of *k* “*cuts”* (like cutting a deck of cards) and then taking the top ballot. Sampling with *k-*cuts works well with ballot-polling audits, but doesn’t work at all for ballot-comparison audits (which need to find a ballot with a specified imprinted ballot ID).

We also propose the use of *Bayesian audits*for determining whether to accept the reported election outcome or to continue the audit (by examining a larger sample). Bayesian audits are an alternative to “*risk-limiting audits,*”and are of particular interest when no risk-limiting audit method is available or feasible.

# Sampling with *k-*cut

How can one pick a ballot “at random” from a given stack of ballots?

The usual method is to generate a random ballot number (using cryptographic methods), and then to count down in the stack to the ballot at that position. This method is tedious and error-prone when the stack is large.

Our proposed alternative, *k-*cut, works as follows to randomly sample a single ballot from a stack of ballots. (This procedure can be repeated to sample multiple ballots.)

* Pick a suitable small integer *k* (we suggest using *k = 6*).
* Perform *k* “cuts,” where each cut removes a random fraction of ballots from the top of the stack and places those ballots at the bottom of the stack.
* Pick the ballot now at the top of the stack as your selected ballot.

Although each cut may individually be slightly non-uniform, repeating the operation *k* times smooths out the statistics to give acceptably uniform results.

A detailed analysis of the *k-*cut method appears in Mayuri Sridhar’s Master’s thesis (2019), and in (Rivest and Sridhar, 2018). Further research is underway to show how to improve (decrease) *k,* by making use of randomness “hints” when picking a cut size; decreasing *k*  would provide further efficiency improvements.

The *k-*cut method has been used successfully in several pilot election audits (Indiana, Michigan, Rhode Island); going forward it is an attractive choice for use in actual (ballot-polling) audits.

# Bayesian tabulation audits

A ballot-level statistical post-election tabulation audit keeps drawing cast paper ballots and manually examining them, until it is determined that the sample drawn provides sufficient support for the reported outcome, or until all cast paper ballots have been examined.

There is more than one way to use statistical methods to define a “stopping rule” for the audit. “Risk-limiting audits” are one way; Bayesian audits are another (although there is some overlap).

A risk-limiting audit asks “What is the current risk if we stop the audit now?”, and stops the audit if this risk is below a pre-defined *risk limit*. Here risk is defined as the (conditional) probability that if the reported outcome is incorrect that the audit would accept the reported outcome as correct.

A Bayesian audit asks “What is the `upset probability’?” – the probability that examining all of the cast paper ballots would show the winner to be different than the reported outcome – and stops the audit if this upset probability is below a pre-defined *upset probability limit.*  Bayesian methods are used to define the upset probability as the posterior probability of an upset, given the sample and given a prior probability on ballots.

These definitions appear very close, but there are nonetheless significant differences.

For one thing, risk may be viewed as a *worst-case* definition, while upset probabilities are more of an *average-case* definition. Given the adversarial nature of elections, a risk-limiting audit may in general be a more appropriate choice than a Bayesian audit (and we recommend using risk-limiting audits whenever possible).

Also, risk and upset probabilities appear not to be on the same scale: a risk-limit of five percent may correspond (roughly) to an upset probability limit of one-half of one percent or so (ten times smaller). Determining the relationship between risk and upset probability is an active research area. Achieving risk below a certain risk-limit is not the same thing as achieving an upset probability below a certain upset probability limit. One can’t naively switch back and forth between the two models; the definitions mean different things.

Nonetheless, risk-limiting audits and Bayesian audits are *highly compatible.*  Their high-level structure is identical: drawing increasingly large samples until a stopping rule says to stop. A Bayesian audit can easily “piggy-back” on a risk-limiting audit, using the same sample data, and computing upset probabilities while the risk-limiting audit is computing risk. This can provide additional comfort and confirmation that the reported outcome is likely to be correct.

How does one implement a Bayesian audit? The following outline sketches one approach (based on “Polya’s Urn”) for computing an upset probability:

1. Draw an initial random sample of the cast paper ballots; examine each sampled ballot manually to determine the voter’s intent.
2. “Pretend” to examine the remaining ballots, but instead of drawing new ballots randomly to examine manually, look at randomly chosen *previously examined* ballots again (with probability proportional to the number of times each ballot has been previously examined).
3. Compute the winner of the set of all (really drawn and pretend-drawn) ballots.
4. Repeat steps 2—3 many times. The fraction of time that the reported winner loses is the “upset probability”.

The Bayesian audit stopping rule says to stop the audit if the estimated upset probability is below the pre-defined upset probability limit.

The Bayesian method is quite simple. One nice feature is that it works at the ballot-level, and is independent of the voting method used. The same approach works for plurality, approval voting, instant-runoff voting, etc. All that is needed is a method to determine the winner (step 3 above) for a set of ballots, and one must have such a procedure anyway just to run an election!

It should be noted that Bayesian methods require the definition and use of a “*prior probability distribution”* giving the assumed likelihood of seeing any particular ballot prior to seeing any sample data. In this use of Bayesian methods for post-election audits, defining such a prior is much easier than for many other applications of Bayesian methods, since the only purpose of the prior here is to ensure that *a priori* all ballot choices are judged equally likely. The prior is weighted to ensure that it “steps out of the way” when the sample data arrives. In the above sketch, a typical prior would be effected by including one extra ballot for each candidate in the sample as part of step 1.

One may also easily extend Bayesian methods to handle ballot-level comparison audits, or various forms of stratified audits (where some strata are ballot-polling and some are ballot-comparison). Details omitted here; see Rivest (2018) for an expanded treatment, and see the original Rivest and Shen (2012) paper for more variations.

Bayesian methods have been implemented and used in various pilot audits; typically as a “free add-on” to a risk-limiting audit. For example, in the December 2018 pilot audit of a proposition on the ballot in Rochester Hills, Michigan, Kellie Ottoboni and Philip Stark computed (for a sample of 76 ballots with 50 Yes votes and 26 No votes) a risk of 2.1%, while Mayuri Sridhar computed an upset probability of 0.3%.

Again, these numbers are not directly comparable, but both are significantly below their pre-defined limits, so both the risk-limiting audit and the concurrent Bayesian audit (on the same sample data) confirmed the reported election outcome. See (Brennan, 2019) for details. Bayesian audits have been used in a number of other pilot audits as well.

In summary, Bayesian methods provide additional tools in the auditor’s arsenal, and may in some cases (for complex voting methods where no risk-limiting audit method is known) be the only tools available. For typical plurality elections, Bayesian methods are probably best as a possible concurrent “second opinion” on the correctness of the reported election outcome.

# References

Scott Borling, Tina Barton, Chris Swope, Virginia Vander Roest, Mayuri Sridhar, Kellie Ottoboni, Liz Howard. Eds. Ron Rivest, Jerome Lovato, Philip Stark. A Review of Robust Post-Election Audits: Various Methods of Risk-Limiting Audits and Bayesian Audits. Brennan Center for Justice and Verified Voting. 2019 (to appear).

Ronald L. Rivest. Bayesian Tabulation Audits: Explained and Extended. arXiv <https://arxiv.org/abs/1801.00528> (2018-01-02)

Ronald L. Rivest and Emily Shen. A Bayesian Method for Auditing Elections. Proceedings 2012 EVT/WOTE Conference. <https://www.usenix.org/conference/evtwote12/workshop-program/presentation/rivest>

Mayuri Sridhar. Optimization for Election Tabulation Auditing. MIT EECS Master’s Thesis. February 2019. <https://mayuri95.github.io/main.pdf>

Mayuri Sridhar and Ronald L. Rivest*. k*-cut: A Simple Approximately-Uniform Method for Sampling Ballots in Post-Election Audits. Proceedings Financial Cryptography, February 2019, Fourth Workshop on Advances in Secure Voting. <https://fc19.ifca.ai/voting/program.html>