Using Barcoding to Assist Manual Recounts of Touch Screen Voting Machines

Jason Sclar  
Department of Computer Science  
Yale University  
jason.sclar@yale.edu

May 8, 2006

Abstract
I study the problem of using technology to aid in the recounting of elections. The contradiction is that as election processes become more automated there is less public trust in their reliability and more desire to see them recounted. However, manual recounts are slow and expensive in terms of resources. The problem with an automated recount is that it must be shown to be utterly infallible because the desire to recount an election comes from the lack of public trust in automated voting processes. The goal is to come up with a way of barcoding ballots to make the public believe in a recount that is conducted based on the barcodes. I use the open-source datamatrix barcode. I examine its limitations. Ultimately, I propose how it can be used in our election processes.

1 Introduction
Starting with the 2000 presidential election, the integrity of voting systems has become an issue in which the public has taken an interest. The last two presidential elections have been marred by recounts and charges of faulty voting machines altering outcomes in several states. While the best known problems, the well-known “hanging chads” and the infamous “butterfly ballot” were not caused by automated voting systems, the problems far more likely to persist in future elections were caused by touch screen voting machines, or DREs. The biggest problem with these systems is that they are essentially a black box. They produce no paper trail, so the only record of how votes were actually
cast is the reported outcome at the end of the day. The fact that the CEO of the major producer of DREs, Diebold, guaranteed he would help George W. Bush win Ohio in 2004, coupled with the company’s refusal to release their source code has only fueled suspicion of fraud in these machines. However, public faith could be restored by a simple paper trail system that would permit a separate recount.

2 Paper Trail

A good paper trail must contain two parts. The first is a visual record that can be read by the human eye and the second is a representation of the first part as a barcode. The first serves two purposes. The first has more to do with reassuring the public than with the actual recount. It offers the possibility of a second recount done by hand. While these are expensive in terms of time, the option of having one is important in terms of trust. If the receipts only contain the barcode, that makes them no better than the current black box machines in terms of transparency. While, hopefully, the manufacturer is honest, they still have the option of acting maliciously by taking the vote and then tallying it however they desire and printing out a corresponding barcode. Giving the voter an opportunity to see a printed receipt where they can see their slate of candidates in print should provide enough security for even the most paranoid voter. The second purpose of printing text on the receipt is to allow the person scanning it to correctly orient it. While datamatrix can be read from any angle, if the person reading it knows the orientation, this allows for an extra round of error correction that can make the barcode more robust.

There are two potential types of errors that can occur for these paper trails. The first are caused by natural wear and tear on the receipts. This includes creasing, ink smudges and tears in the paper. The second, and much more problematic is a malicious
adversary, one who creates errors by either ripping, crumpling or drawing on codes. This adversary should not be acquainted with datamatrix to the point that he can willfully make any code unreadable.

3 Datamatrix

For the barcode on the receipt, I chose to use datamatrix [1]. Datamatrix is a two-dimensional, error correcting code that can, under optimal conditions, be properly read when printed as small as several millimeters, having lots of distortion and erasure in the code itself. It is a binary code made up of small black and white squares, which represent 1s and 0s. All datamatrix codes have an even number of bits by an even number of bits. Almost all are square, though there are a small number of rectangular configurations. The barcode itself consists of two parts, the border and the center. The border is the same for all datamatrix codes. The left and bottom edges are made up of entirely black squares and the top and right edges alternate black and white squares, so the top-right square in all datamatrix codes is white. The purpose of the border is two-fold, to properly orient the code and to help show the reader the size of the squares being read. Because of this border, datamatrix codes can be read from any angle.

The other key thing to the functioning of datamatrix is the redundancy in the codes. This is created using the Reed-Solomon algorithm [2]. The Reed-Solomon algorithm can add any number, t, of parity bits to the data. When decoding the data, the number of correctable errors is directly related to the number of bits of parity used in creating the code. A decoder can correct up to t/2 symbol errors, where bits are deliberately wrong, or t erasure errors, where bits are unknown. The computational complexity of encoding and decoding using Reed-Solomon is directly related to the
number of bits of parity being used in the particular code. If we denote the number of
symbol errors as $s$ and the number of erasure errors as $r$, a Reed-Solomon code will
always properly decode provided that $2s + r < t$. Otherwise there are two possibilities.
The first is that the decoder will determine that the original message cannot be recovered
from the code. The second is that the decoder will recover an incorrect message and
determine that it is correct. This is based on the distribution of the errors within the code.
Reed-Solomon works better on burst errors, where the errors are all in sequence in the
code. These are more likely to produce an unreadable code; whereas a wider distribution
of errors throughout the code is more likely to produce a misread code.

Of the two errors, having the barcode be unreadable is vastly preferable to having
it be misread. If the barcode is unreadable, it will say as much and that one ballot will
have to be manually counted. However, if a barcode is misread, it is a disaster. Misread
barcodes will not set off any warning flags because the reader will believe they have been
accurately read. In this case, it can take the entire recount to realize that there were
misread codes and that a manual recount is necessary. Any number of these errors, even
an extremely small one, would cause the automated recount to be worthless and create a
need for a time consuming manual recount. Therefore, the goal is to have all barcodes be
either readable or determined to be totally unreadable.

Because of the high cost of using the Galois’ Fields necessary for the Reed-
Solomon encoding underneath datamatrix codes, there are accepted standards for how
many bits can be encoded to a barcode of a given size and how many errors it can
withstand and still be read. For the purposes of encoding election results, the most
practical datamatrix format is a grid of 16x16, 18x18 or 20x20. These have a capacity of
24, 36 and 44 digits, respectively. The most practical string for to create and encode for an election allots a number to each election with 0 representing an abstention and then each candidate being assigned a number from 1-9. If a race has more than 9 candidates, it may be allotted a second digit. While almost all elections can be represented in this way on the 16x16 grid, the advantage to encoding to a larger grid is that larger grids can handle more errors and are more robust for reading. The 16x16 grid can be properly read with up to 6 symbol errors or 9 erasure errors. The 20x20 grid can be read with up to 9 symbol errors or 15 erasure errors.

3 The Limitations of Datamatrix for a Paper Trail

For my simulations, I created 20x20 codes. Because the code was already properly oriented after it had been produced, I was able to correct any errors to the border independent of any error-correcting scheme. Knowing the orientation and being able to correct the border before any attempts at reading the code adds to the robustness of the datamatrix barcode. Being able to correct potentially wrong bits without having them count against the fairly low number of permitted errors greatly improves the decoder’s ability to properly decode altered codes. For a 20x20 barcode, it raises the number of correctable symbol errors to a maximum of 85, though this does include the stipulation that 76 of those errors are in the border of the code. Also, if the orientation is unknown and the border gets significantly altered, it can render the whole code unreadable because the reader may not be able to identify the proper rotation for the code. Because the receipts that would be produced contain text at the top, this allows for proper orientation and correction of the border before the codes are decoded, as in the simulation.
4 Results

Ultimately, however, the computational complexity of decoding datamatrix prevents it from being as viable a technology for receipts as I had previously hoped. While it is exceptionally robust, even when scaling a 144x144 grid down to millimeters, it does not appear well suited to stop a malicious user. The low number of correctible errors that any given barcode can sustain, while seemingly high enough for natural wear and tear are not high enough to work successfully when there is a malicious user deliberately changing large blocks of bar codes. With my source code, I ran tests where I altered each bit with some set probability (set in mutate.h as THRESHOLD). Once this got over 3%, my results for being able to successfully read the codes fell off drastically. By 10%, almost no ballots were ever considered readable.

While this appears to be fine for natural errors, I do not believe this would successfully handle a malicious adversary who is changing even moderate sized blocks of the barcode. An adversary, with a black magic marker, coloring in twenty-four non-edge, white squares could render most codes unreadable. One way to prevent this would be to leave the readable list of the slate elected off of the receipt. This creates three problems. The first is that the receipt no longer tells the orientation of the barcode. Now it cannot fix the border before it attempts to read the barcode. This lowers the number of acceptable natural errors. The second problem is that ballots deemed to be unreadable now could not be manually counted, so this undermines the ability to carry out an accurate recount. The third, and final, problem is again one of public faith. If a voter cannot see the names of the candidates they voted for on their receipt, then the system is still no better than the present black box systems.
5 Future Work

The next step with testing this system is to actually print out hard copies of the receipt files and see how accurately they can be read after undergoing both natural and deliberate wear and tear. I suspect that an adversary creating deliberate problems with the receipt will be able to stop them from being counted properly. This can be tested with a relatively small amount of work though a cost of nearly $1000. There are several programs available that take user input and turn them into image files of datamatrix barcodes that conform to the ISO standard. These cost between $150-300. There are also a small number two-dimensional barcode scanners and decoders available starting at around $500. Armed with these two products, a person would be able to produce actual receipt files, subject them to various stresses, such as smudging, tearing and drawing on them, and see how well they can be read after that. This has to be done before datamatrix can either be considered or abandoned as a solution to this problem.
6 References


[3] Datamatrix Font and Encoder,

    http://www.barcodestore.com/software/idautomation/datamatrix/specs.html