Tamper-Evident, History-Independent, Subliminal-Free Data Structures on PROM Storage -or- How to Store Ballots on a Voting Machine (Extended Abstract)

David Molnar, UC Berkeley
Tadayoshi Kohno, UC San Diego
Naveen Sastry, UC Berkeley
David Wagner, UC Berkeley

Abstract

We enumerate requirements and give constructions for the vote storage unit of an electronic voting machine. In this application, the record of votes must survive even an unexpected failure of the machine; hence the data structure should be durable. *At the same time, the order in which votes are cast must be hidden to protect the privacy of voters, so the data structure should be* history-independent. *Adversaries may try to surreptitiously add or delete votes from the storage unit after the election has concluded, so the storage should be* tamper-evident. *Finally, we must guard against an adversarial voting machine's attempts to mark ballots through the representation of the data structure, so we desire a* subliminal-free representation. *We leverage the properties of Programmable Read Only Memory (PROM), a special kind of write-once storage medium, to meet these requirements. We give constructions for data structures on PROM storage that simultaneously satisfy all our desired properties. Our techniques can significantly reduce the need to verify code running on a voting machine.*

1 Introduction

"Good" vote storage mechanisms are a critical enabling technology for reliable and secure electronic voting. Without a "good" vote storage mechanism, votes might be lost during power failures, an adversary might be able to undetectably tamper with the voting record post-election, or an adversary with access to the voting record might be able to compromise voter privacy. Unfortunately, "good" vote storage in practice has traditionally not been easy to obtain, either because of a lack of understanding about what the appropriate goals should be, as in the case of Diebold [6], or because of genuine subtleties in the design of a "good" vote storage mechanism.

Because of the subtleties with and importance of "good" vote storage mechanisms, we consider it an important research objective to thoroughly (A) investigate what "good" vote storage actually means, hence the quotes around "good" above, and (B) investigate how to build "good" vote storage mechanisms in practice.

Defining "good." We believe a vote storage mechanism should have at least the following seven properties:

1. *Simple* : We desire a vote storage mechanism that is simple to implement, analyze, and verify.
2. *Reliable* : The vote storage mechanism should not rely on fragile moving parts or other components that might fail during use.
3. *Durable* : The record of votes should survive unexpected crashes of the vote storage mechanism.
4. *Tamper-evident* : Anyone with read access to the voting record should be able to detect post-election tampering.
5. *History-independent* : Assuming a *non-malicious* vote storage mechanism, the contents of the voting record should not reveal information about the order in which ballots were cast.
6. *Subliminal-free* : A *malicious* vote storage mechanism should not be able to undetectably embed covert information into the voting record.
7. *Cost effective* : Election officials may only deploy these solutions if the cost per voter is not significantly more expensive than alternative technologies.

Durability is important because, even if the vote storage mechanism is reliable, catastrophic events like power loss and battery failure might cause a machine to crash. History independence is important since it might otherwise be possible to compromise voter privacy if one also knows the order in which people voted [6]. Subliminal freedom is important because, if the property is not met, a malicious vote storage device could use the covert channel to leak information about who voted for whom [5]. Although a voter-verifiable paper audit trail (VVPAT) might alleviate some of the need for these properties, we still consider history-independence and the absence of subliminal channels to be very important if VVPAT-enabled electronic voting machines also maintain digital copies of the voting record.

Our proposals for simultaneously achieving these properties use a combination of hardware and algorithmic techniques. We summarize our proposals for simplicity, reliability, tamper-evidence, history-independence, subliminal-freeness, and durability next, with details in the full paper [10]. We evaluate the cost effectiveness of our techniques in Section 4.

Scope of our work. Our work focuses on the interaction between voting machine *software* and voting storage. As we describe below in our discussion of an architecture for an electronic voting machine, our adversarial model includes malicious software in the voting machine that wishes to undetectably tamper with or leak information about already cast votes. Our key insight is that by properly designing the vote storage unit, we can reduce what must be verified about the main code of the electronic voting machine. For example, if the vote storage unit provides tamper-evidence, we need not verify that the rest of the software refrains from overwriting previously cast ballots. We believe our methods culd help reduce the cost to verify voting machine software.

We assume that other mechanisms will be used to ensure the physical security of the polling place and of the voting hardware. Physical security is essential. For instance, if the storage unit can be physically swapped with an "identical-looking" unit carrying different vote totals, then the integrity of the entire election process is compromised. Likewise, an adversary with physical access to the polling place could plant hidden video cameras, mount timing or power consumption attacks, or attempt to tamper with voting equipment. While important, we do not address these threats in this work; they are outside the scope of this paper.

2 Our Architecture for an Electronic Voting System

We now describe our proposed architecture for a Direct Recording Electronic (DRE) voting machine (see Figure 1). We decompose the machine into three parts: a main *DRE* component, a *vote storage module*, and a removable *storage* device, represented in the figure by a PROM. A PROM is a special type of storage in which bits may transition from 1 to 0 but not vice versa. If traditional antifuse [15] PROMs are not readily available, we suggest that it might be possible to instead use One-Time Programmable Electrically Programmable Read Only Memory (OTP EPROMs). An EPROM is a device in which bits may be set electronically, but may only be cleared by exposure to ultraviolet light. An OTP EPROM is an EPROM device in an opaque housing that prevents such erasure. Using an OTP EPROM is a relaxation of the write-once property because OTP EPROMs can still be reset under some circumstances, e.g., if their protective coating is removed and if they are exposed to ultraviolet light. OTP EPROMs are reasonably inexpensive, so we can afford to use each chip only once. For instance, the ST Microelectronics M27C4001-10B1 is a 4MBit OTP EPROM chip that costs $2.75 [9].

At the beginning of the election, the DRE component sends an Initialize command to the vote storage module to open the polls. Thereafter, the DRE component handles all interaction with the voter and produces a cast ballot. The ballot is in turn passed to the vote storage module by calling Insert. The vote storage module writes the ballot to the storage device, using the Clearbit and Readbit interfaces. At the end of the election, the DRE invokes the Finalize operation, which tells the vote storage module to modify the storage device in such a way as to preclude further votes from being recorded. The storage device might then be removed from the voting machine and transported securely to election headquarters, and the stored ballots will be available for tallying through the storage device's Dump interface.

Security assumptions. We assume that the DRE has no access to writable media except through the vote storage module. Furthermore, we assume that none of the DRE's state survives between voting sessions. We also assume that no information can pass from the storage device to the stateless DRE component. This ensures that the only long-term storage available to the DRE is managed by the vote storage module. Our intent is that the vote storage module is a small and easily-verified piece of code that is isolated from the main part of the DRE. We assume for the purposes of this paper that the ballot reflects exactly the intent of the voter; for example, a voter-verified paper audit trail may be used to check the machine's result. Other than these restrictions, however, the DRE component may behave arbitrarily and adversarially.

3 Approach

Tamper-evident vote storage. Our approach for providing tamper-evidence involves what we believe to be a novel application of Manchester codes and programmable read-only memory (PROM). At a high level, we apply a Manchester code to votes before writing the votes to a PROM. This allows us to exploit the fact that bits on a PROM can only

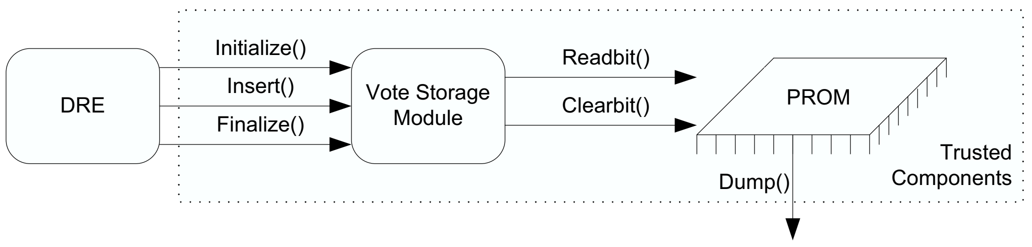

Figure 1. Our DRE architecture. The main part of the DRE has no access to writable media except through the vote storage module. After each vote is cast, the completed ballot is passed to a special vote storage module that arranges for it to be recorded on permanent storage.

transition from 1s to 0s and not vice versa. The Manchester encoding of a n-bit string x is a $2n$-bit codeword $M(x)$ obtained by applying the mapping $0 \mapsto 01$, $1 \mapsto 10$ to each bit of x. For example, the Manchester encoding of the string 101 is 100110.

The key property of the Manchester encoding is that if any set of 1 bits in a Manchester codeword are flipped to 0s, then the result is no longer a valid codeword. For example, if the first 1 bit is cleared in 100110, the resulting string 000110 is no longer a valid Manchester codeword. This gives us a simple test to determine whether an adversary has tampered with data stored on a PROM: we encode the data using the Manchester encoding, and then later check that the PROM still contains a valid codeword. By the properties of the PROM, any modification by the adversary will result in an invalid codeword and so can be easily detected. Consequently, Manchester encoding ensures that data, once written to the PROM, cannot be undetectably changed. In the full paper we show an extension of this idea that uses only $n + \lceil \lg(n+1) \rceil$ bits instead of the $2n$ bits required for a Manchester encoding. We use PROMs and avoid other forms of write-once media, such as CD-Rs, because of our simplicity and reliability goals.

History-independent and subliminal-free vote storage. We are careful to ensure that one can implement our history-independent and subliminal-free data structures on a PROM, which is *not* the case with other proposals for history-independence that assume erasable storage [1, 3, 8, 11]. Even under the restriction of non-erasable storage, there are several natural approaches for achieving history-independence that use random coins when storing each vote: for example, to store a vote, select at random an unused location on the PROM and store the vote there. However, because a malicious vote storage device could use its choice of "random" coins to leak information to a conspirator [5], and because one might prefer not to assume the presence of a secure random number generator even in the absence of malicious intent, it is also natural to look for history-independent data structures that are deterministic.

Suppose Ω is the set of items which one may insert into our data structure. If Ω is finite and fairly small, one can create a history-independent data structure by preallocating a region of a PROM for each element in Ω and, within each region, counting in unary the number of times that element is inserted.

When Ω is large, however, the unary counter approach becomes very space inefficient. Leveraging an observation by Naor and Teague [11] that the lexicographic ordering of a set of elements must by definition be history-independent, one can adopt the following "copyover list" data structure for large Ω: to insert an element into the data structure, sort the new element along with the existing elements, write that sorted list to the next unwritten portion of the PROM, and then erase the previous sorted list by setting all its bits to zero. Notice that the length of the just-erased list depends only on the number of items in the data structure, not their content or the order in which they were inserted. Notice also that the copyover list supports arbitrary write-in candidates, subject to a bound on the length of the candidate's name.

We can improve our space efficiency by either combining the above two approaches or by employing a hash table, though in the latter case we must take special care to ensure history-independence even when two elements hash to the same bucket. The key insight we use to ensure this property is to implement each bucket using another history-independent data structure, such as the previous "copyover list" proposal. The main improvement in space efficiency for such a hash table is that copying only occurs when there is a hash collision. Storing each bucket as a separate copyover list yields a data structure we call a *lexicographic chain table*.

The resulting structure is history-independent. Just as in the case of a single list, the length and position of

Data Structure	Requirements	HI	TE	VSF	Space
Unary Counter	$\|\Omega\|$ small	Yes	Yes	Yes	$O(n)$
Copyover List	None	Yes	Yes	Yes	$O(n^2)$
Lexicographic Table	None	Yes	Yes	Yes	$O(n \log^2 n)$ w.h.p
Random Table	RNG	Yes	Yes	(1)	$O(n)$

Figure 2. Summary of our data structures and their properties. Here RNG stands for "random number generator." HI, TE, and VSF stand for history-independence, tamper-evidence, and verifiable subliminal-freedom, respectively. Random placement tables are subliminal-free only if the RNG has been verified to be perfect.

each bucket's copyover list depends only on the number of items in the list. This leaves the issue of whether different amounts of deleted material in each bucket leak information about the order of votes. To see that this is not the case, observe that the amount of deleted material depends only on the number of items in each bucket, which is equal to the number of hash collisions for that bucket. The adversary, however, can compute the number of collisions itself given only the abstract contents of the data structure and the hash function (independent of the order in which items were inserted). Therefore, this does not leak information about the order in which votes were cast. We provide subliminal-freeness using an idea from collusion-free protocols: the voting machine commits to the random hash key before it sees any votes, then reveals the hash key at the end of the election [7]. Details and extensions are in the full paper [10].

Another approach we consider is a *random placement table,* in which we pick a random position in a table when an element is inserted. If the position is empty, the element insertion succeeds and the position is filled, otherwise another position is picked. The main benefit of a random placement table is that it is space-efficient. The main drawback, however, is that the random placement table requires a random number generator that has to be verified correct; otherwise a malicious DRE could mark ballots or otherwise leak information in the order of placement. We give a comparison of our data structures in Figure 2.

Engineering issues with PROM storage. We acknowledge that there are engineering challenges with implementing our solutions on PROMs. For example, the atomicity properties of PROMs vary widely; writes may occur out of order or may only partially complete, especially in cases where loss of power or a crash occurrs during a write. We do not consider these issues in detail here, but do note these issues have been addressed by others in other contexts. For example, Niijima [12] reports on a log-structured file system designed to achieve good performance and durability even in the face of atomicity limitations. Further afield, but also illustrating both the limitations of and work-arounds for similar technologies, Gal and Toledo survey a wide variety of file systems designed for flash memories that use "wear-levelling" to avoid writing too many times to a single storage sub-unit between full resets [2].

In general, this work does not cover these engineering issues, focusing instead on the algorithmic design of data structures for vote storage. We consider PROM storage as an arbitrary-length array addressable at the bit granularity, with atomic operations to read and set each bit. Furthermore, we assume that all operations complete one at a time, in the order in which they are issued. Dealing with the engineering issues of PROM devices requires future work. We note, however, that our data structures do not require the same flexibility as a general-purpose file system. In addition, the performance requirements of the vote storage application are fairly modest.

Caveat. While we envision a vote storage mechanism meeting our properties being an important component of electronic voting machines, vote storage is only one of many components that need to be made reliable and secure. Having addressed vote storage, we hope others can address the remaining portions of electronic voting machines.

4 A Sample Cost Analysis

We now calculate the cost for implementations of our ballot data structures. We use data from the November 2, 2004 election for precinct 213100001 in Alameda County, California[1]. Our dataset is from the Smart Voter website at `http://www.smartvoter.org`. The storage requirements, and hence cost of storage, depends on the number of voters, number of candidates, and number of write-ins.

There were 9 races with a total of 31 candidates and 20 (yes-no) state propositions or local ballot measures. In the absence of accurate polling place data, let us derive an estimate for the number of voters using each voting machine. If we assume it takes 15 seconds to consider each race or bal-

[1]Precinct 213100001 is the precinct for the Claremont Hotel, site of the IEEE Symposium on Security and Privacy.

lot measure over 13 hours of polling time, this allows for a maximum of 104 voters. For safety, we triple that guess and assume there are a maximum of 300 voters per voting machine. In contrast, anecdotal evidence suggests that a busy machine might receive 100–150 votes at most.

We use a unary counter to represent each race or option in a measure. For the example ballot, we will use $31 + 2 \times 20 = 71$ unary counters, one for every choice. Recall that incrementing a unary counter uses one bit. We must size the unary counter to be larger than the maximum number of expected voters; in our case, a total of $71 \times 300/8 \approx 2.60$ KB suffices.

Of course, for each of the 9 races, the voter is free to write in the candidate of their choice. A write-in consists of the candidate's name (30 characters long) as well as a unary counter ($300/8 \approx 38$ bytes) for a total of 68 bytes per distinct write-in. We store each race in a separate data structure, so a write-in for a candidate in one race cannot be linked to any other votes in that race or any other race. This provides subliminal-freedom even in the presence of write-ins.

Note that the number of distinct write-in candidate names (ignoring multiplicity) is likely to be significantly less than the total number of write-in votes (including multiplicity), since several people may write in the same candidate. Obtaining data on the number of distinct write-ins in actual elections is tricky, though. Published data for this election indicates a 0.6% write-in rate for each of the local races. The data, however, is incomplete. In particular, no data was available on write-ins in national races; also, the data available to us did not provide the number of distinct write-ins. If we make the assumption that the write-in rate is uniform across all DREs, we would expect each race to have fewer than 0.6% distinct write-ins. Election administrators must be prepared, however, for significantly more write-ins.

In Figure 3 we display a graph comparing three of our constructions[2]. The cost is almost entirely dependent on the maximum number of distinct write-ins per race that election officials need to be prepared for: the greater the desired write-in capacity, the more chips the DREs need. In comparison, the cost for storing regular candidates (or repeated write-in votes for a single candidate) is negligible. Using random placement tables, the entire election can fit on a single 4MBit chip, even with 50 distinct write-ins per race. If election workers use a copyover list instead, only two such chips are required to achieve the same capacity. For our cost estimates, we use the 4MBit OTP EPROM chip described in Section 2 at a cost of $2.75 per chip. Notice that this provides a tremendous degree of over-capacity:

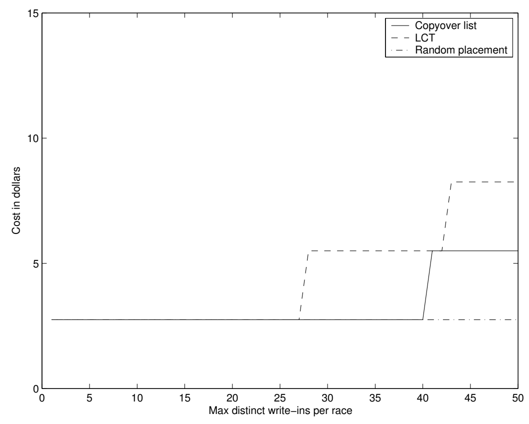

Figure 3. Cost estimates for a single voting machine. Cost is a step function because we assume that storage must be purchased in increments of a 4 megabit chip. Note the expected number of distinct write-ins is less than 1 per race, so costs remain reasonable even if the actual number of write-ins far exceeds the expected number.

absent a denial of service attack, it would be rare to encounter a race that received 50 write-ins (which corresponds to half the expected number of voters), let alone 50 distinct write-in names. Tamper-evidence is provided by writing a Manchester-encoded checksum of the chip as a whole at the end of the election, as we detail in the full version, and so requires only a small amount of overhead, which we factor into our calculations.

Even though the copyover list exhibits quadratic asymptotic behavior, it performs well in the voting application. Lexicographic hash chaining, even though it features better asymptotics, must over-provision space for many distinct values hashing to the same value; to obtain a suitably low chance of exhausting the space in one hash bucket induces a high constant value that is hidden in the asymptotic behavior.

The costs for using the random placement tables or a copyover list compare favorably with the cost of an optical scan ballot. Optical scan ballots can cost $0.10 to $0.30 per voter [14]; securely storing ballots on a DRE using our secure vote storage techniques cost less than $0.05 per voter.

5 Related Work

Micciancio defined the basic problem of privacy for data structures and gave a construction of "oblivious trees" [8]. Naor and Teague extended the notion and gave several constructions, including the priority hash chaining construction

[2]With the lexicographic chaining table, there is a small chance of bucket overflow. Therefore, for this scheme, we choose table and bucket sizes so that the probability of bucket overflow is no more than 2^{-30}.

that we build on [11]. Hartline et al. improved several data structures and studied relaxations of the history independence requirement [3]. Irani, Naor, and Rubinfeld showed that a Turing Machine with write-once polynomial space decides exactly the languages in the class P [4]; our PROM model differs in that multiple writes are allowed, so long as the new value can be obtained by flipping 1s to 0s. Shamos independently proposed using a hash table with unary counters for storing votes in a history-independent manner [13], although Shamos's scheme does not achieve perfect history independence. There is also a large body of work on implementing file systems with solid state storage, such as flash memory. Gal and Toledo survey this work, and Niijima gives an in-depth report on a flash memory file system that addresses engineering issues we discuss above [12, 2].

6 Conclusions

We have described constructions for data structures suitable for vote storage on electronic voting machines. Our techniques exploit commodity hardware, so that the consumables cost for an election is reasonable. Our vote storage module provides a history-independent, tamper-evident, durable, and subliminal-free representation method to securely record ballots in an isolated module. We believe that a vote storage module with these properties will simplify the design of DRE voting systems and ultimately enhance the public's confidence in them.

7 Acknowledgments

We thank David Jefferson, Susan Hohenberger, abhi shelat, and Adam Stubblefield for helpful discussions and suggestions. We also thank the anonymous reviewers of the IEEE Symposium on Security and Privacy and Philippe Golle. D. Molnar was supported in part by NSF CCR-0325311 and by a NSF Graduate Research Fellowship. T. Kohno was supported by NSF CCR-0208842, NSF ANR-0129617, and NSF CCR-0093337, and performed part of this research while visiting the University of California at Berkeley. N. Sastry was supported by NSF CNS-0524252 and by the Knight Foundation under a subcontract through the Caltech/MIT Voting Technology Project. D. Wagner was supported by NSF CCR-0093337 and CNS-0524252. We finally thank They Might Be Giants for their song entitled "Subliminal."

References

[1] N. Buchbinder and E. Petrank. Lower and upper bounds on obtaining history independence. In D. Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 445–462. Springer-Verlag, Berlin, Germany, 2003.

[2] E. Gal and S. Toledo. Algorithms and data structures for flash memories. *ACM Computing Surveys*, 37(2):138–163, 2005.

[3] J. D. Hartline, E. S. Hong, A. E. Mohr, W. R. Pentney, and E. Rocke. Characterizing history independent data structures. In *ISAAC '02: Proceedings of the 13th International Symposium on Algorithms and Computation*, pages 229–240. Springer-Verlag, 2002.

[4] S. Irani, M. Naor, and R. Rubinfeld. On the time and space complexity of computation using write-once memory or is pen really much worse than pencil? *Mathematical Systems Theory*, 25(2):141–159, 1992.

[5] C. Karlof, N. Sastry, and D. Wagner. Cryptographic voting protocols: A systems perspective. In *Proceedings of the 14th USENIX Security Symposium*, pages 33–49. USENIX Association, Aug. 2005.

[6] T. Kohno, A. Stubblefield, A. D. Rubin, and D. S. Wallach. Analysis of an electronic voting system. In *IEEE Symposium on Security and Privacy*, pages 27–40. IEEE Computer Society, May 2004.

[7] M. Lepinksi, S. Micali, and a. shelat. Collusion-free protocols. In *STOC '05: Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing*, pages 543–552. ACM Press, 2005.

[8] D. Micciancio. Oblivious data structures: applications to cryptography. In *STOC '97: Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 456–464. ACM Press, 1997.

[9] S. Microelectronics. M27C4001-10B1 OTP EPROM 4MBit data sheet, 2005.

[10] D. Molnar, T. Kohno, N. Sastry, and D. Wagner. History-independent, tamper-evident, subliminal-free data structures on PROM storage -or- how to store ballots on a voting machine, 2006. Cryptology eprint archive report 2006/081.

[11] M. Naor and V. Teague. Anti-presistence: history independent data structures. In *STOC '01: Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing*, pages 492–501. ACM Press, 2001.

[12] H. Niijima. Algorithms and data structures for flash memories. *IBM Journal of Research and Development*, 39(5), September 1995.

[13] M. Shamos, March 2005. Post to VSPR mailing list. https://lists.csail.mit.edu/pipermail/vspr-wg-ut/2005-March/000061.html% .

[14] VotersUnite.org. Comparing annual costs of DRE and optical scan systems, 2005. http://www.votersunite.org/info/ComparingAnnualCostsDREvPBOS.pdf.

[15] Wikipedia. Antifuse. Visited February 17, 2006. http://en.wikipedia.org/wiki/Antifuse .