Clash attacks and the STAR-Vote system

Olivier Pereira and Dan S. Wallach

Université catholique de Louvain and Rice University

Abstract. STAR-Vote is an end-to-end cryptographic voting system that produces both plaintext paper ballots and encrypted electronic records of each ballot. We describe how clash attacks against STAR-Vote could weaken its security guarantees: corrupt voting terminals could identify voters with identical ballot preferences and print identical receipts for them, while generating electronic ballot ciphertexts for other candidates. Each voter would then be able to "verify" their ballot on the public bulletin board, but the electronic tally would include alternative ciphertexts corresponding to the duplicate voters. We describe how this threat can be exploited and mitigated with existing STAR-Vote mechanisms, including STAR-Vote's use of Benaloh challenges and a cryptographic hash chain. We also describe how this threat can be mitigated through statistical sampling of the printed paper ballots as an extension to the risk-limiting audits that STAR-Vote already requires.

1 Introduction

Clash attacks, a term coined by Küsters, Truderung and Vogt [12], are a family of attacks on verifiable voting systems in which corrupted voting machines manage to provide the same vote receipt to multiple voters, so that the verification procedure succeeds for each voter individually, while corrupted voting machines are able to cast whatever vote they like for each of the voters who were given a duplicate receipt. Examples of clash attacks have been proposed against ThreeBallot [15], Wombat [3], and a variant of the Helios voting system [1].

Clash attacks happen when voting machines can prepare ballots in such a way that a voter cannot verify that they contain an element that is unique to them. This is the case for STAR-Vote [2], since a voter will not have seen any other ballots, and thus won't know that ballot ID numbers are reused.

How would a clash attack on STAR-Vote appear in practice? Under the assumption that the software running inside one or more STAR-Vote voting stations was corrupt, the voting station could detect when a voter casts an *identical* ballot to a previous voter. At this point, the voting station would print a paper ballot corresponding to the previous voter while potentially having the freedom to generate a ciphertext ballot completely unrelated to the voter's intent.

Each of these voters now has a receipt that includes the hash of a completely valid vote for exactly each voter's intent. Unfortunately, the two receipts are pointing to same exact ballot, which neither voter would necessarily discover, while meanwhile a fraudulent ballot would be counted as part of the electronic

tally. STAR-Vote provides evidence to auditors both while the election is ongoing and after it has completed that could *potentially* detect clash attacks like this, but suitable procedures were not part of the original STAR-Vote design. In this paper, we describe several variants on clash attacks and present a number of countermeasures to discover or rule out the presence of clash attacks.

2 STAR-Vote

We describe the key elements of STAR-Vote, omitting in various places details that are not relevant for our analysis in this paper.

Entities. Running a STAR-Vote election requires the participation of four groups of persons: (1) *Voters*, who submit votes and are invited to participate in various optional auditing operations as part of the end-to-end (E2E) verifiable component of STAR-Vote; (2) *Internal auditors*, who run the risk limiting audit (RLA) part of STAR-Vote; (3) *Trustees*, who are responsible of holding and using the decryption keys responsible for the confidentiality of the votes; and (4) *Election managers*, who are responsible of setting-up and supervise the election operations.

As part of their role, the election managers need to setup in each voting station a locally networked set of devices: (1) *Voting stations*, to be used by the voters to produce ballots, under electronic and paper format; (2) *Ballot boxes* that receive the paper ballots; (3) a *Ballot control station* (BCS) that orchestrates the various devices in a voting precinct.

Setup Before an election starts, the trustees jointly produce an election public key k_T for a threshold commitment consistent encryption scheme [9], and two unique hash chain seeds are chosen: zp_0 and zi_0, one for the public audit chain, and one for the internal audit chain. The internal audit chain logs, and replicates on all machines connected to the local network, all the events happening in each voting station, ballot box and on the BCS, with a fine-grained modularity, with the intent of collecting as much evidence as possible in case of a disaster (while making sure to encrypt all potentially sensitive data). The public audit chain logs all the elements that are needed in order to run the end-to-end verification of the election, and is designed so that it can be used and verified while only using data that hide the content of the ballots in an information theoretic sense (in particular, this hash chain does not include any encrypted vote). Every time a ballot is printed, the current state of the public chain is printed on the take-home receipt prepared for the voter.

The BCS also selects a public key k_C for an internal use and, through the BCS, all voting stations and ballot boxes are initialized with k_T, k_C, zp_0, zi_0. The final step of the setup is to initialize both chains with a unique identifier of the precinct in which the machines are located.

Casting a ballot When signing-in, a voter receives a unique token t associated to his ballot style bst. When entering the booth, the voter enters his token and the voting station displays an empty ballot of style bst, so that the voter can make his selection v.

The voting station processes these choices as follows, based on the current values zi_{i-1} and zp_{i-1} of the hash chains:

1. It broadcasts the information that the token t introduced by the user is consumed.
2. It computes the number of pages that the printed ballot will take and, for each page, selects an unpredictable ballot page identifier $bpid$. This identifier is not intended to become public or listed anywhere in digital form, for vote privacy reasons that will be explained later, and is used as part of the RLA.
3. It computes an encryption of the vote $c_v = \mathsf{Enc}_{k_T}(v \| zi_{i-1})$ and a vector c_{bpid} of ciphertexts that encrypt $\mathcal{H}(bpid \| r_i)$ with k_T for each race r_i printed on page $bpid$ of the ballot.
4. It selects a unique ballot casting identifier $bcid$ and computes $c^C_{bcid} = \mathsf{Enc}_{k_C}(bcid)$, which will be used by the BCS to detect when a paper ballot corresponding to a given electronic record is cast in a ballot box.
5. It broadcasts a message containing the time t, bst, c^C_{bcid}, c_{bpid}, c_v and computes a hash $zi_i := \mathcal{H}(zi_{i-1} \| t \| bst \| c^C_{bcid} \| c_{bpid} \| c_v)$ for inclusion in the internal audit trail.
6. It prints each page of the ballot, the last page being made of two pieces that can be easily taken apart. The first part contains a human readable summary of v and a machine readable version of $bpid$, $bcid$ and the ballot style bst. The second part is a take home receipt that contains a human readable version of the election description, the time t and $zp_i := \mathcal{H}(zp_{i-1} \| bst \| t \| \mathsf{CExt}(c_v))$. The $\mathsf{CExt}()$ function extracts, from a ciphertext, a component that is a perfectly hiding commitment on the encrypted plaintext. This commitment is expected to include ZK proofs of knowledge on an opening to a valid vote.

When receiving this, the controller decrypts c^C_{bcid} and appends the pair $(bcid, zp_i)$ into a local table, until a ballot with that $bcid$ is scanned by a ballot box.

Challenging a voting station When a voter, or a local auditor, wants to challenge the voting station, she brings the printed ballot to a pollworker. The pollworker: (1) stamps the ballot to mark it as spoiled; (2) scans the $bcid$ so that the ballot is recorded in the internal and external hash chains to be treated as part of the spoiled ballot box.

Later, at tallying time, the spoiled ballots are all decrypted (or their randomness is disclosed by the voting station that produced them) and they are posted on the election bulletin board for public verification (including by the voter).

Casting a ballot If the voter is happy with the ballot printed by the voting station, it brings it to a ballot box. There, the two pieces of the last ballot page are split, the take-home receipt is kept by the voter, and all the pages are put into the ballot box, which scans the $bcid$ printed on each page, and both hash

chains are then appended with the information that these pages have been cast and that the corresponding encrypted votes need to be included in the tally. If the scanned *bcid* is unknown to the BCS, the ballot is rejected by the ballot box and an error signal is triggered.

Electronic tallying At the end of the day, all the encrypted votes c_v's that have been marked as to be included for the tally are checked for validity and aggregated into an encryption $\mathbf{c_v}$ of the tally. This tally is then jointly decrypted by the trustees and published. (This is done as needed for the different races, ballot styles, ...)

Then, $\mathsf{CExt}()$ is applied to all the c_v's, the result is published with all the information needed to check the *zp* hash chain, and the trustees publish a proof that the tally is consistent with *zp*. Eventually, the trustees jointly and verifiably decrypt and publish the content of the spoiled ballots.

Audit of the electronic process Anyone can perform a number of verifications from the published information: (1) check the validity of the published $\mathsf{CExt}(c_v)$'s; (2) check that the tally is consistent with the published $\mathsf{CExt}(c_v)$'s; (3) check the validity of the *zp* hash chain; (4) check the number of ballots against the number of voters if the information is public; (5) check that the scanned spoiled ballots were correctly built.

Furthermore, voters are invited to check whether the zp_i value printed on their receipt appears in the list of ballots included in the tally and, if they spoiled a ballot, to check that their spoiled ballot really appears in the list of spoiled ballots. If any of these verification steps fails, complaints should be filed.

Audit of the paper ballots After having checked the validity of all the encrypted votes, the trustees supervise (or perform), contest by contest, a shuffle of all (c_{bpid}, c_v) pairs corresponding to valid ballots (after splitting the c_v's into race components), yielding a list of (c'_{bpid}, c'_v) pairs. This shuffle needs to be made verifiable, at least with respect to the privacy properties that it offers, either by using a fully verifiable mix-net [11, 17], or possibly by using a lighter solution like a marked mix-net [13].

After completion of this shuffle, the trustees decrypt all c'_v and c'_{bpid} tuples. This decryption yields, for each race r_i, a list that contains $\mathcal{H}(bpid \| r_i)$ and the cleartext choices that should be those on the ballot page *bpid* for race r_i. This table is made available to all the people who are taking part to the risk-limiting audit. The use of the hash function and high entropy *bpid*'s guarantees that noone is able to decide which race results belong to the same ballot page, which helps defeating pattern voting attacks.

From this table, a ballot-comparison risk limiting audit (RLA) can take place. The gist of the process is to start by verifying that all the hashes in the above-computed table are unique, that the number of such hashes is consistent with the number of ballots and their ballot styles as reported for the ballot boxes, then

repeat, a number of times that is a function of the election margins computed from the results of the electronic tally, a process that consists in: (1) selecting a random ballot page (2) read its *bpid* and search for $\mathcal{H}(bpid \| r_i)$ values in the table for all races r_i present on the ballot; (3) compare the corresponding plaintexts to the paper ballot.

3 Clash attacks on STAR-Vote

We now present a threat model for how a clash attacker might be able to operate and discuss how clash attacks might be detected.

3.1 Threat model

Clash attacks require a fairly sophisticated attacker, capable of running malicious code on every computer in a given STAR-Vote precinct: the controller, every ballot terminal, and the ballot box as well. Under normal circumstances, we might hope that this is not feasible, but certainly many commercial voting systems have suffered from vulnerabilities that allowed for the viral spread of malware (see, e.g., the results of California's "Top to Bottom Review" [7] and Ohio's "EVEREST" [14] studies in 2007). Consequently, under such an attack, many of STAR-Vote's security protections become weaker, but others remain strong.

- STAR-Vote specifies human-readable paper ballots, printed by its voting stations, and deposited in a ballot box. It remains possible to ignore the electronic results entirely and tally the paper ballots independently, whether by hand or by scanning into another computer.
- STAR-Vote specifies the use of Benaloh challenges [5, 6] to catch a voting machine in the act if it tries to substitute a ciphertext that doesn't correspond to the voter's intent, as printed on the plaintext ballot. Our attacker will try to tamper with unchallenged ballots, and will try to take advantage of the end-of-day distribution of STAR-Vote's encrypted ballot records.
- STAR-Vote specifies a SOBA risk-limiting audit [4], which selects electronic ballots at random and requires the audit to identify the corresponding paper ballots. If this audit selects a printed ballot for which there is no corresponding electronic record, then the audit will discover this absence.
- STAR-Vote encrypted ballots are constructed from homomorphically encrypted counters which include non-interactive zero knowledge (NIZK) proofs that they are well-formed (e.g., no counter indicates anything other than one or zero votes for a given candidate). Our attacker does not have the power to forge these proofs.
- STAR-Vote specifies the use of a cryptographic hash chain to preserve the integrity of the encrypted ballots. Every voter is also given a printed receipt containing the hash of the record of their vote, which in turn includes the hash of the previous record. While we cannot guarantee that voters will verify

every single receipt, any voter receipt protects the integrity of every vote cast before it in the same precinct. Our attacker does not have the power to find hash collisions and thus cannot create alternative histories consistent with each voter's hash.

Consequently, it's within the scope of our threat model for a STAR-Vote voting terminal, when given a voter who expresses selections identical with a previous voter, to print a duplicate copy of the prior voter's ballot and receipt, while publishing an encrypted vote for other candidates STAR-Vote's hash chain. This paper analyzes the ways in which such a powerful adversary might attempt to operate and how it might later be discovered.

Our threat model does not empower an attacker to tamper with *every computer in the world*, merely every computer in the control of a given local election authority. External computers might still be considered trustworthy. As an example, a smartphone app that scans ballot receipts and posts them to an independent cloud service for analysis could be considered beyond the reach of our attacker.

3.2 How could this work on STAR-Vote?

A clash attack on STAR-Vote could happen in different ways, based on the following approach:

1. Alice comes and expresses a vote v, encrypted as c_v^A, and included in the hash chain, leading to a public hash zp^A, which is printed on the ballot with v, *bpid* and *bcid*. The paper ballot is split and cast.
2. Bob comes and happens to express the same vote v, something that is noticed by the malicious voting station. The voting station then produces a ciphertext c_v^B, encrypting a different vote v^* (of the same style), and encrypts $bcid^*$'s and $bpid^*$.
3. When printing Bob's paper ballot, the voting station prints a ballot with v written on it, the hash zp^A that was printed for Alice, and $bcid^*$'s and $bpid^*$.

The expectation is that, when Alice and Bob read their paper ballot, they see their vote intent correctly reflected and, at the end of the day, they will both find a ballot containing the expected hash in the public hash chain: they will both look at the same place. However, the cheating machines manage to replace Bob's vote v with a different vote, while not modifying the total number of votes.

Several variants of this attack can be considered, depending on whether $bcid = bcid^*$ and $bpid = bpid^*$. Various strategies can also be adopted when voting stations want to scale the attack: they can create many pairs of clashing ballots, each pair having a distinct hash, or create one large clash, in which many ballots would have the same hash, or adopt any strategy in between.

3.3 Can we detect it?

The high-level description of STAR-Vote, as reflected above and in the STAR-Vote documentation [2, 16], does not seem to provide obvious ways of spotting

the attack that we just described. We split our analysis according to the two parts of the verification of a STAR-Vote election: the end-to-end electronic verification part, and the RLA part.

End-to-end verification. On the side of the electronic process, all the mandatory verification steps succeed: the trustees tally the expected number of ballots with the expected races, the hash chains looks legitimate, and the voters find their zp_i on the election bulletin board. However, if the verification is pushed further and ballots are challenged, then discrepancies can be detected.

If Bob decides to challenge its voting station, the voting station can offer a decryption of c_v^A, which will be consistent with the printed voter intent and be included in the hash chains. Inspections can also be made regarding the *bcid* and *bpid* (though they do not seem to be explicitly prescribed in the original documentation).

The internal hash chain contains an encryption c_{bcid}^C of the ballot casting identifier *bcid* that is printed on the ballot. Here, the attacker has two options:

1. It can generate a fresh *bcid* for Bob's ballot. In this case, the printed ballots will have distinct *bcid*'s, as expected, and a possibly honest ballot box or BCS has no way of detecting a potential duplicate. But, if Bob's ballot is spoiled, there will be a discrepancy between the *bcid* printed on the ballot, and the one pointed by the hash printed on the receipt, which will be Alice's *bcid*. So, if the voting station bets that Bob's ballot will not be spoiled, no evidence is left (at the *bcid* level, at least).
2. It can resuse Alice's *bcid* on Bob's ballot. In this case, the decryption of the encrypted *bcid* pointed by the hash printed on the receipt will be consistent with the *bcid* printed on the paper ballot. But the ballot box will contain two ballots with identical *bcid*'s. So, by adopting this strategy, the voting station can pass an inspection of the *bcid* at spoiling time, but it will leave (potentially hard to find) evidences in the ballot box. Also, if the BCS happens to be honest, it may happen that it notices the same *bcid* coming twice.

With the current description of STAR-Vote, and given the above threat model, the second strategy seems likely to be a successful one, at the electronic level at least.

A deeper inspection of the hash chains will show other discrepancies, though: every time a ballot is cast or spoiled, this event must be recorded in both hash chains. There are again two attack strategies that can be followed here, as depicted in Figures 1 and 2.

1. As depicted in Fig. 1, the BCS may mark Alice's ballot as cast in both chains as soon as it is notified of Alice's dropping of her ballot in the ballot box. If Bob cast's his ballot, then the BCS marks the malicious ciphertexts prepared on Bob's behave as cast too, and nothing is visible. However, if Bob decides to spoil his ballot, then the machines are facing a difficulty: the public hash chain should have Alice's ballot marked as spoiled, but this creates an inconsistency in the chain since this ballot has already been marked as cast. So, a public evidence is left, and this one is easy to notice.

2. As depicted in Fig. 2, the BCS can record that Alice cast her ballot, but not append that information in the hash chains, and wait to see what Bob will do with his ballot. Now, if Bob casts or spoils his ballot, the BCS can simply append that instruction in the chain (and always mark Bob's ciphertext as cast, in order to preserve a consitency in the number of ballots cast and spoiled). However, if Bob spoils his ballot, Alice's ballot will be shown as spoiled on the bulletin board, and Alice may file on complaint on the ground that she cast her ballot.

Still, this last strategy seems to be the "safest" for malicious machines: a discrepancy will only become visible if Bob challenges his ballot and if Alice checks her ballot on the election board, notices the problem, and files a complaint.

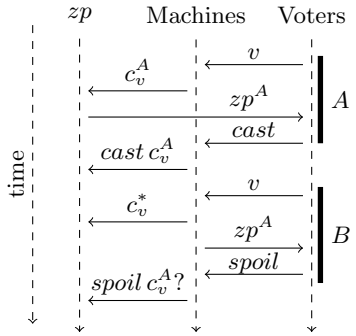

Fig. 1. Clash attack with immediate recording of ballot casting.

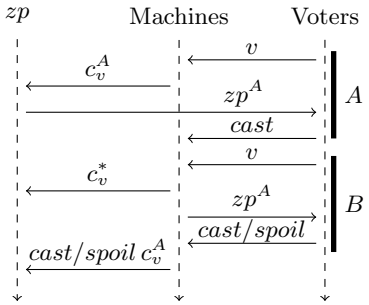

Fig. 2. Clash attack with recording of cast ballots delayed.

To conclude, it appears that all clash attack variants can be detected by the audit trail left by the end-to-end verifiable part of STAR-Vote. Our analysis sheds a new light on the importance of the "cast" and "spoil" marks that are included in the hash chains, and stresses that the system should provide easy mechanisms to detect that no single ballot appears as cast or spoiled in the bulletin board. (If the cast and spoiled ballots appear separately, as it is done in many end-to-end verifiable voting system prototypes, this may be easily overlooked.)

Taking a look back, a countermeasure against clash attacks proposed by Küsters et al. [12] consists in asking voters to type a random number and to print that number on the receipt, for verification. This would be feasible with STAR-Vote and would render duplicate ballots very difficult to produce, as duplicates could only happen between voters with identical vote intents and picking choosing the same random number. However, we would prefer to avoid extra interaction with the user. As a variant, we imagine that receipts could add a sequence number, synchronized across every voting station, printed in large type so it's visible to poll workers and watchers. Any repeats of recent numbers or incidences of numbers wildly out of sequence would be suspicious.

Risk limiting audit. The risk limiting audit component of STAR-Vote is expected to offer confidence in the election results, independently of the effectiveness of the end-to-end verifiable component. The inputs of the RLA are:

1. Ballot manifests, that list all ballot boxes, as well as the number of ballots contained in each box, and the style of these ballots.
2. An electronic vote record, that contains, race by race, a list of hashes of the form $\mathcal{H}(bpid \| r_i)$, and associated to each of these hashes, the choices made for race r_i that must be printed on the ballot page *bpid*.
3. Paper ballots in boxes, as per the ballot manifests.

For the sake of our discussion, we assume that the manifests are correct, and that the election outcome resulting from the electronic record is consistent with the one announced from the end-to-end verifiable tally.

The *bcid*'s are not part of the electronic records, and are therefore not used in the RLA. The *bpid*'s, though, offer the crucial link that is expected to define the bijection between paper ballots and electronic records. Again, there are two possible strategies for malicious machines running a clash attack:

1. A malicious voting station can print different *bpid*'s on the ballots with clashing receipts. In this case, if Bob's ballot is selected as part of the RLA, the *bpid* printed there won't point to the electronic record of Alice's ballot, which is precisely the kind of discrepancy that the RLA is designed to efficiently detect.
2. A malicious voting station can print identical *bpid*'s on Alice's and Bob's ballots. Assuming that both ballots are cast, the boxes now contains two ballots with the same *bpid*. And, even if one of the two ballots is picked as part of the RLA, no discrepancy will appear: Alice's electronic record will be picked in both cases, and will match the paper ballot content.

This second strategy seems to be a successful one: the RLA assumes that there is a bijection between the paper and electronic ballots, and does not attempt at verifying that there is a bijection indeed. In order to solve this issue, we investigate the possibility of a bijection audit.

4 Bijection audit

We want to determine whether there is a bijection (i.e., a one-to-one correspondence) from the paper ballots to the electronic ballots. Paper ballots that do not have a match in the electronic records are easy to detect. However, bijection failures resulting from clashing ballots can only be detected if we pick duplicate paper ballots. Of course, as described earlier, two voters might well have identical voting selections, but every ballot page is *supposed* to have a unique *bpid*, which is a randomly selected 128-bit number, and thus highly unlikely to repeat (from the birthday paradox, this will only become likeley after casting around 2^{64} ballots in the same box). The discovery of two identical *bpid* numbers on two separate pages would imply election fraud. For the remainder of this section, we will assume that we want an auditing procedure that's completely independent of the end-to-end verifiable side of STAR-Vote. We don't want to rely on the

hash chains, the cryptographic receipts, or the Benaloh challenges. We wish to design a process for validating the bijection by considering the paper ballots and the cleartext electronic vote records, alone. We assume that the shuffle and audit data are kept at the precinct level, so that inter-precinct clashes would be equivalent to missing ballots.

4.1 Why not sort?

A seemingly attractive solution is to sort the ballots by *bpid*, after which detecting duplicates would be a simple linear scan. The problem is that we're dealing with as many as $N = 1000$ paper ballots in a given precinct. We need a completely manual process that a small set of poll workers can accomplish quickly. Manual sorting doesn't scale well.

A merge sort, wherein the pile of ballots is partitioned into smaller piles, each of which is sorted, and then the sorted piles are merged, might seem attractive. The initial partition happens quickly, giving a hypothetical sixteen poll workers 1/16 of the ballots. If our workers sorted their initial piles using an insertion sort, taking 10 seconds per ballot, then the initial phase would only take ten minutes. The merging phase, however, would be more cumbersome. If we followed a tree-like binary merging process, each merge phase must consider twice as many ballots and would use half as many poll workers. Again, assuming ten seconds per ballot, the first phase would reduce 16 to 8 piles in 21 minutes. The second phase would reduce 8 to 4 piles in 42 minutes, then 4 to 2 piles in 84 minutes, with the final merge taking 168 minutes. The whole process totals up to almost 5.5 hours. Even if our poll workers can insert a ballot every 5 seconds instead of 10 seconds, this process might still take 3 hours.

Of course, there are many variations, but they all suffer from expensive phases. A bucket sort, for example, requires a linear scan to begin, partitioning the ballots based on their prefixes, but it makes the merging process trivial, since the sorted buckets can simply be stacked rather than painstakingly merged.

If we lived in the 1960's, we might suggest the use of a *sorting machine*, such as were used with punchcard decks [8]. Alas, such devices now only exist in museums, with any modern need to sort pieces of paper being handled digitally after the use of a high-speed scanner. We need a procedure that can be accomplished without the use of computers, and this procedure must only take a few minutes, not hours. In return, we're willing to trade off a *guarantee* of finding a duplicate for a chosen *probability* of that detection.

4.2 Audit methodology

The SOBA risk limiting audit [4] is designed to provide a required degree of confidence in its outcome, regardless of the number of ballots. We will now specify a simple sampling procedure that can audit a pile of ballots for uniqueness of the *bpid* numbers, assuming that the *bpid*'s are actually random and that at most one duplicate is made of any given *bpid*. (We will relax this assumption in Section 4.3.)

Variable	Definition	Example value
m	margin of victory (fraction)	0.05
d	duplicated ballots (fraction)	0.03
n	number of ballots sampled	100
N	total number of ballots in the box	1000
P_d	probability of discovering duplicates	0.95
T	number of trials (e.g., precincts sampled)	5

Subsampling the ballots. We will start with N ballots (perhaps as many as 1000 in a box) and need an efficient procedure for subsampling a more reasonable number n (perhaps 50, perhaps 100), with the added concern that our adversary will be aware of our subsampling methodology. We imagine that our poll workers can roll dice to select specific digits for use in a search. (Since digits are printed in hexadecimal, a 16-sided dice would be most convenient.) We have a variety of options for how to proceed. For example, to sample $\frac{1}{16} \cdot \frac{1}{16} = \frac{1}{256}$, we can roll dice to select specific values for the first and second digit of the *bpid*. To sample $\frac{1}{128}$ of the ballots, we could pick two possible values for one of the digits (i.e., $\frac{1}{16} \cdot \frac{2}{16} = \frac{1}{128}$).

In this fashion, we can design samples to get close to any ratio that we might want. For example, if we truly want to sample exactly 10% of the ballots, we might select three possible values for the first digit and nine possible values for the second digit, yielding roughly $\frac{3}{16} \cdot \frac{9}{16} = 0.1055$. So long as the resulting fraction is slightly larger than the target ratio of $\frac{n}{N}$, we will have the number of samples that we want. It doesn't matter if the adversary knows the digit locations we will consider (e.g., most-significant vs. least-significant). If there exists duplicate *bpid* numbers anywhere in the pile, then they have a chance of being selected by the sample (i.e., we are not making n random draws from the pile of ballots; we are making queries against *bpid* digits). Conversely, and anticipating on our further discussion, it's important that we roll dice for the specific values of the digits. Otherwise, the adversary could guarantee that the *bpid* values on duplicate ballots were never selected for an audit.

Discovering duplicates. Once we have our sample of ballots, we then must discover duplicates in the sample. If the sample is small enough, sorting is going to be much more feasible. For example, ballots could be split into piles based on the most-significant-digit of *bpid* and then each pile could be sorted by hand. This process would take minutes, not hours. But what are the odds of discovering a duplicate? We can solve for the probability of discovery and then rearrange the equation to solve for the fraction f of the *bpid*'s to be sampled. The first line below expresses that the probability of detection P_d equals one minus the probability that all Nd *bpid*'s which are duplicates are not picked, which will happen with probability $1 - f$ every time.

$$P_d = 1 - (1 - f)^{Nd}$$
$$f = 1 - (1 - P_d)^{1/Nd}$$

Of these, it's helpful to use the equation for f and plug in values we might expect for d, P_d and N. For example, if $d = 0.03$, $N = 1000$, and we want $P_d = 0.95$, then $n = 95$. With a sample of 95 ballots, we can thus have a 95% chance of discovering a duplicate. Here are some other solutions:

d	P_d	f	n
0.010	0.95	0.259	259
0.030	0.95	0.095	95
0.050	0.95	0.058	58
0.100	0.95	0.03	30

If the duplication rate d is high, we can detect it with a fairly small number of samples n and a very high probability of success P_d. However, we can see that we need more than 250 samples when d is only 1%. So when might poll workers be required to conduct such a large sample? Consider that every process like this occurs after the election is complete, which means that we know the margin of victory m. We can simply specify that $d = m$, i.e., we're looking for enough ballot duplication to change the election outcome. Consequently, as the margin of victory shrinks, only then do we need to sample a large number of ballots.

Repeated trials. Consider what might happen if we repeated the above process across multiple precincts, selected at random. It's entirely possible, from the attacker's perspective, that they could just as well attack one precinct or attack every precinct, so as an auditor, we should look at more precincts. Or, if we simply want to avoid the non-linear costs of manually sorting large numbers of paper ballots, we could conduct multiple trials in the same precinct. The resulting $AggregateP_d = 1 - (1 - P_d)^T$, simply multiplying together the odds that the attacker gets away with it in each trial.

d	P_d	n	T	$AggregateP_d$
0.010	0.60	88	1	0.60
0.005	0.40	97	1	0.40
0.010	0.60	88	5	0.99
0.010	0.50	67	5	0.97
0.005	0.40	97	5	0.92

Now, even with very small duplicate rates like $d = 0.01$, we can conduct five trials, perhaps across five precincts or perhaps within the same precinct, of only 67 ballots per trial. While each trial has only a 50% chance of discovering a duplicate, all five together have a 97% chance. (Sorting 67 ballots, five times, is significantly easier than sorting 259 ballots, even once.)

4.3 Non-random duplicates

Next, we will consider the possibility that an attacker arranges for every duplicate ballot in the box to share the same *bpid*. In this case, the odds of detection

are only the odds that the dice match the attacker's *bpid*. If we match, then we get every duplicate. If we fail to match, then we get no duplicates.

Furthermore, in our threat model, the attacker can control the *bpid* number distributions, making sure that any biases introduced through the duplicates is evened out over the other ballots. For example, if the duplicates were more likely to have a "3" in the first digit, the attacker could arrange for other ballots to never start with a "3", and could go further and arrange for "9" to occur most often. Consequently, we cannot rely on relatively simple procedures, like splitting on digits and counting each pile, as a statistic to detect duplicates.

Instead, we will propose a sampling methodology with a relatively low success rate, in any given precinct, but which will gain its power in aggregate when repeated across many precincts. We will only assume that we can make a random draw of n ballots from any given ballot box. Rather than this process involving dice, we instead imagine a process similar to "cutting" a deck of cards, whereby each draw involves splitting a pile of ballots and selecting the next ballot from the location of the cut.

Given this sample, we can then manually sort it and look for duplicates. If n is, for example, 100 ballots, this process will only take a few minutes. The odds of successfully detecting duplicates are a function of the size of the sample n and of the fraction of duplicates d. We compute this by measuring the probability of selecting *only* from the non-duplicates and the probability of selecting *exactly one* of the duplicates: $P_d = 1 - (1-d)^n - n \cdot (1-d)^{n-1} \cdot d$

If $d = 0.01$ and $n = 100$, then P_d is approximately 26%[1]. If this is repeated for T trials, we can compute $AggregateP_d$ in the same fashion. For example, with $T = 10$ trials, we again can find a precinct with duplicates with a 95% probability. This represents significantly more work than we needed in the case with randomly distributed duplicates, but it's still feasible to conduct this without requiring hours of effort.

(We note that *sampling without replacement* would be preferable, both because it would slightly increase the odds of success, and because we wish to physically demonstrate the existence two separate ballots with the same *bpid*. The equation above, however, assumes *sampling with replacement*, which is only an approximation that becomes less accurate when N gets smaller. An accurate combinatorial expression of P_d is not particularly necessary for our discussion.)

4.4 Non-random precinct corruption

We first considered uniformly distributed duplicates within a precinct. We next considered how every duplicate in a precinct could share the same *bpid*, making them harder to find via sampling. Here, We apply the same consideration to the election as a whole. We now assume that our attacker wants to do all of the corruption in a very small number of precincts rather than spreading it uniformly out across every precinct.

[1] If we select values for n and d where $n \cdot d = 1$, then the expression for P_d tends to $1 - 2/e$.

Let's revisit P_d and $AggregateP_d$ from above. In the limiting case where *every* ballot in a precinct is a duplicate, then *any* audit that touches more than one ballot will detect the duplication. This means that P_d is either trivially 1 or 0. A similarly process we can conduct in every precinct might be to draw a handful of ballots and eyeball them for duplicate *bpid* numbers. This would guarantee the detection of a precinct with 100% duplicates.

4.5 Linear auditing with buckets

The subsampling methods described above all begin with a linear pass to select ballots having IDs with a desired pattern. This section presents an alternative method for detecting duplicates that requires only two linear passes over the ballots.

This method requires some basic record-keeping that can be accomplished with pencil and paper. In the first pass, we will be mapping from ballot IDs to *buckets*. Let's say we use the first two hex digits of the *bpid*, which we can map to a 16x16 grid, pre-printed on a single sheet of paper; a poll worker would then write down the third (and maybe fourth) hex digit in the bucket. At the end of the pass, the buckets are searched for duplicates. If the number of ballots and buckets are well chosen, the number of ballots per bucket will be small, and this search will be easy. If a collision is found, the bucket is marked as suspicious: this may come from a collision on the first hex digits that will stop after a few more digits, or be the result of a clash. The purpose of the second pass is to inspect the suspicious buckets: during that pass, the ballots belonging to these buckets are further inspected, in order to make sure that no clash happens.

An exact estimation of the expected number of ballots per buckets is challenging to express: these are non-trivial variations around the "birthday paradox" problem. However, fairly accurate approximations based on the Poisson distribution can be obtained (see, e.g., DasGupta [10]).

Let us say that we want to estimate the probability $P(b, n, k, m)$ that, in a setting with b buckets and n ballots, there are k buckets containing m ballots. We first compute the probability that m ballots with randomly selected *bpid* would go into the same bucket: that probability is b^{1-m}. Now, we consider the process of selecting n ballots as actually picking $\binom{n}{m}$ m-tuples of ballots. This is of course an approximation, since the independence between these m-tuples is not obtained when we just have a pile of n ballots, but it turns out that it is accurate enough for our purpose (it over-estimates the number of collisions, while being asymptotically exact). The last step consists in estimating the probability $P(b, n, k, m)$ as the probability that an event happening with probability $\lambda = \binom{n}{m}/b^{1-m}$ happens k times, as given by the Poisson probability mass function: $P(b, n, k, m) = e^{-\lambda}\frac{\lambda^k}{k!}$.

Let us consider two examples: one in which we apply this approach to around 100 ballots, as would occur in the non-random duplicate search mechanism of Sec. 4.3 for instance, and one in which we apply this approach to a full box of around 1000 ballots.

Linear search of duplicates among 100 ballots. Let us consider that we have a 16x16 grid on a single tabloid format page, providing 256 buckets, and that each bucket is split into 3 blocks in which 3 hex characters can be written (the page would be large enough to offer blocks of 8x15mm, which is comfortable).

Based on the expression above, there is a probability 0.5 that at most 19 buckets will contain two (or more) ballots, and 0.95 that at most 27 buckets contain two (or more) ballots (see Table 1). A small number (less than 5) buckets will contain 3 ballots, and it is most likely (80% probability) that no bucket would take 4 ballots. If this happens, then a separate note could be created at the bottom of the page, in order to compensate for the lack of space. So, all the ballots are expected to fit easily on the grid that we just described.

Now, we can estimate the probability that a collision happens inside a bucket, that is, that two ballots share identical 5 first digits. Here, there is a 0.995 probability that no such collision would happen. In the unlikely case that one happens, then a second linear search is performed in order to determine whether a clash has been detected. As we can see, this procedure is extremely effective.

Ballots	Buckets	Multiplicity	50%	95%
100	256	2	19	27
100	256	3	2	5
100	256	4	0	1
100	256×16^3	2	0	0
1000	1024	5	7	12
1000	1024	6	1	3
1000	1024	7	0	1
1000	1024×16	2	30	40
1000	1024×16	3	0	2

Table 1. Estimation of bucket fillings. The last two columns indicate bounds on the number of buckets containing at least "multiplicity" ballots, bounds that are satisfied with probability 50% and 95% respectively.

Linear search of duplicates among 1000 ballots. Let us now consider that we have four tabloid format pages, each having a 16x16 grid providing 256 buckets, and that each bucket is split into 6 blocks in which 1 hex character can be written. Let us also consider that the first character of the *bpid* is chosen among 4 values instead of 16 (this could be just by prefixing the *bpid* with an extra random symbol).

Now, during the linear pass, four poll workers hold one page each. An other audit officer (possibly under surveillance) makes a linear pass on the ballots, reads the first digit of the *bpid* in order to point to one of the four poll workers holding the grids, then reads the next two hex digit in order to point to one bucket, and finally read the next hex digit to be written in that bucket.

Based on the Poisson estimate, it is fairly unlikely that a single bucket will need to contain more than 6 ballots: this would happen with probability 0.16, and just one or two buckets will contain exactly 6 ballots (again, see Table 1). If we turn to the number of collisions that will be found on the single hex digit written on the bucket, we can expect that around 30 buckets will contain a single collision, and that it is quite unlikely to observe more than a 2-collision in a single bucket.

In order to sort these collisions out, we make a second linear pass on all the ballots, but only focusing on the collisions. The four officers take a fresh grid, mark the colliding buckets and the prefixes that need to be examined, and now write 3 more hex digits in the bucket when a suspected ballot is read (there will be enough space, since we only write something down for the few colliding ballots). Any collision repeating on these extra digits would be an overwhelming indication of a clash.

4.6 Other potential uses of bijection audits

The assumption of a bijection is at the core of comparison audit processes like SOBA. Our work raises the question of whether bijection audits would be useful to detect clash attacks in other circumstances that could be completely independent of STAR-Vote or even of end-to-end verifiable systems. For instance, in locations where paper ballots have a serial number and paper ballots are scanned in order to perform an electronic tally, ballots with clashing serial numbers could be distributed to voters who are known to vote in the same way (e.g., straight party), and a malicious scanner could replace the images of those paper ballots with clashing serial numbers with fresh ballots of its choice. This would break the bijection from the paper and electronic records, and potentially make a risk limiting audit ineffective, unless a bijection audit is run first.

5 Recommendations and conclusions

STAR-Vote has a variety of security mechanisms and we've described a number of different auditing and testing procedures. This section considers how these individual procedures and tests might best be combined to defeat clash attacks.

Real-time receipt auditing. The bijection-audit procedures described in the previous section are feasible, but are considerably more expensive than a SOBA audit, so it would be helpful to have a cheaper alternative.

Recall that a clash attack would cause the receipts of a significant number of voters to be exactly the same. As such, we propose that independent poll watchers, or perhaps the official poll workers themselves, use an independent electronic tool to sample these receipts as they go by. This could be implemented with a smartphone app that scans a printed QRcode, provided that a comparison between the result of the scan and the printed value is made. If the same value

is ever scanned twice, then either a ballot receipt was accidentally scanned twice or a duplicate was produced.

One nice aspect of this procedure is that we can rely on independent computers, outside the influence of our attacker, to simplify the process. The odds of successfully detecting a duplicate are the same as with the audit procedure we described in Section 4.3, only without the requirement for sorting the sampled ballots. This makes the procedure easy to perform. And because the ballot receipts are safe to share with the world, this procedure can be performed by anybody. Of course, if a duplicate is ever discovered, suitable alarms should be raised and a more invasive audit conducted.

We note that this process would be easy to perform across every precinct in an election, making it particularly valuable for detecting focused attacks on a small number of precincts as described in Section 4.4. Also, as described in Section 3.3, Benaloh challenges may discover clash attacks in real-time, provided that the public hash chain is inspected on the fly, and an attackers that aims for multiple clashes on a single receipt will be more easily spotted than an adversary focusing on mere duplicates, since a single challenged ballot among $n+1$ clashing receipts will make it possible for n voters to see their ballot unduly marked as spoiled on the bulletin board. We discuss how to resolve these issues below.

Post-election ballot auditing. In Section 4.2, we described a subsampling audit process based on digits selected by rolling dice. This is a relatively efficient procedure, but local poll workers might be unwilling to perform it, or might introduce errors by performing it poorly. Also, it's preferable to know the margin of victory for the election, which can be used to select an appropriate number of samples to achieve a desired level of confidence. This won't be possible until the election is complete, so it's probably better to wait until all the ballots are brought back from the local precincts to the election headquarters. The bijection audit procedure could then be performed centrally, on a subset of precincts, alongside the SOBA audits that STAR-Vote already requires.

SOBA risk-limiting audits will sample ballots from across an entire election, while our bijection audits happen at the level of a local precinct. This suggests that the two audits could be conducted concurrently, although it might be procedurally simpler to first conduct the SOBA audit, since it's fast. The bijection audit will be slower, although it's amenable to parallelization in that each precinct can be audited independently.

If the bijection audit fails, this invalidates one of the assumptions behind SOBA, which assumes there is a bijection. Similarly, if post-election verification of the hash chains on the public bulletin board turn up discrepancies (see Section 3.3), we must again resolve these discrepancies.

What if a duplicate is found? If a precinct *fails* its bijection audit or if independent auditors discover duplicate receipts, we now have compelling evidence that a clash attack has occurred. Now, the local election official will be under pressure from all sides. Lawsuits will be filed. Reporters will be asking hard

questions. It's essential to have clear procedures to resolve the conflict. Under our definition of a clash attack, duplicates appear in the paper ballots, but the paper ballots still reflect the intent of the voters, while the ciphertexts are more likely than not fraudulent.

Consequently, faced with this attack, we might discard the encrypted ballots in their entirety and do a manual tally from the paper ballot boxes. This would be slow and would also face the risk that our attacker introduced a small clash attack for precisely the purpose of triggering the fallback to paper ballots, which might as well have been tampered in a coordinated effort. Consequently, we believe an appropriate procedure is to render a judgment on a precinct-by-precinct basis as to whether the paper ballots or electronic ballots are more trustworthy. This judgment would be informed by:

- Conducting a bijection audit and SOBA audit on *every* precinct.
- Considering the available physical evidence (e.g., tamper-evident seals on voting terminals and ballot boxes).
- Auditing the voting terminals for software and/or hardware tampering.
- Auditing the hash chain copies, which should be copied identically across all voting terminals in a precinct.
- Considering other factors outside of the voting system itself (e.g., correlations between different delivery trucks and the confirmed incidences of clash attacks or other election attacks).

STAR-Vote provides multiple forms of evidence of the voters' intent. It's entirely possible, for example, that only a fraction of the voting terminals in a given precinct were tampered, and their hash chains may store a different version of the history of the election. That version of history, for the non-tampered terminals, may be judged worthwhile for the votes cast on those terminals, and then the electronic records might only need to be discarded for the tampered voting terminals. Ultimately, the power of STAR-Vote's design is that it provides election officials with redundant evidence of what happened during the election. We might never anticipate every possible attack, but with STAR-Vote's evidence, we can support a variety of auditing and resolution procedures, enabling the detective work necessary to identify and, if possible, remediate issues.

Concluding thoughts. Clash attacks present a tricky challenge for an election auditor, faced with the possibility of systematic computer tampering. We have shown a number of auditing techniques that can be conducted by poll workers, in a post-election setting, in a tolerable amount of time, mitigating the risk of clash attacks.

Acknowledgements

This work is supported in part by NSF grants CNS-1409401 and CNS-1314492 and by the F.R.S.-FNRS project SeVote. Part of this work was performed when the first author was a Fulbright Scholar at Rice University.

References

1. Adida, B., de Marneffe, O., Pereira, O., Quisquater, J.J.: Electing a university president using open-audit voting: Analysis of real-world use of Helios. In: EVT/-WOTE 2009. Montreal (Aug 2009)
2. Bell, S., Benaloh, J., Byrne, M., DeBeauvoir, D., Eakin, B., Fischer, G., Kortum, P., McBurnett, N., Montoya, J., Parker, M., Pereira, O., Stark, P., Wallach, D., Winn, M.: STAR-Vote: A Secure, Transparent, Auditable, and Reliable Voting System. USENIX JETS 1, 18–37 (Aug 2013)
3. Ben-Nun, J., Farhi, N., Llewellyn, M., Riva, B., Rosen, A., Ta-Shma, A., Wikström, D.: A new implementation of a dual (paper and cryptographic) voting system. In: EVOTE 2012 (Jul 2012)
4. Benaloh, J., Jones, D., Lazarus, E., Lindeman, M., Stark, P.: SOBA: Secrecy-preserving observable ballot-level audits. In: EVT/WOTE '11. USENIX (2011)
5. Benaloh, J.: Simple verifiable elections. In: EVT '06. Vancouver, B.C. (Jun 2006)
6. Benaloh, J.: Ballot casting assurance via voter-initiated poll station auditing. In: EVT '07. Boston, MA (Aug 2007)
7. Bishop, M.: UC Red Team Report of California Secretary of State Top-to-Bottom Voting Systems Review (Jul 2007)
8. da Cruz, F.: IBM card sorters. Columbia University Computing History (Oct 2013), http://www.columbia.edu/cu/computinghistory/sorter.html
9. Cuvelier, E., Pereira, O., Peters, T.: Election verifiability or ballot privacy: Do we need to choose? In: ESORICS'13. pp. 481–498. LNCS, Springer (2013)
10. DasGupta, A.: The matching, birthday and the strong birthday problem: a contemporary review. Journal of Statistical Inference and Planning 130, 377–389 (2004)
11. Jakobsson, M., Juels, A., Rivest, R.L.: Making mix nets robust for electronic voting by randomized partial checking. In: 11th USENIX Security Symposium (2002)
12. Küsters, R., Truderung, T., Vogt, A.: Clash attacks on the verifiability of e-voting systems. In: IEEE Symposium on Security and Privacy (2012)
13. Pereira, O., Rivest, R.: Marked mix-nets. In: Workshop on Advances in Secure Electronic Voting – Voting'17 (2017)
14. Project EVEREST (Evaluation and Validation of Election-Related Equipment, Standards, and Testing): Risk Assessment Study of Ohio Voting Systems (Dec 2007), http://www.sos.state.oh.us/sos/info/everest.aspx
15. Rivest, R.L., Smith, W.D.: Three voting protocols: ThreeBallot, VAV, and Twin. In: EVT'07. Boston, MA (Aug 2007)
16. Travis County Purchasing Office: STAR-Vote: Request for Information for a New Voting System. http://traviscountyclerk.org/eclerk/content/images/pdf_STARVote_2015.06.03_RFI.pdf (Jun 2015)
17. Wikström, D.: A commitment-consistent proof of a shuffle. In: Information Security and Privacy – ACISP. LNCS, vol. 5594, pp. 407–421 (2009)