• *Note: On January 9, 2021 Otto Smith emailed the Chair of LD 24, Bruce Cowan, about vulnerabilities in the voting system planned for use on Jan. 10, 2021 at the Legislative District 24 reorganization. Otto spoke with Mr. Cowan on Jan 10 the day of the reorganization meeting about the vulnerabilities. Discovering more vulnerabilities at the reorganization meeting in his official capacity as a voting observer, Otto prepared this written report. On January 13, 2021 Otto Smith emailed this voting system analysis to the Chair of Legislative District 24, the Chair of Jefferson County Democrats, and the Washington State Democratic Party staff member in charge of technology. On Jan 19, 2021 Otto emailed the document to the Chair of the Washington State Democratic Party. As of Jan. 21, 2021 Otto has received no response and no requests for help in a replacement or a redesign or some other resolution. It is likely Party officials intend to use this flawed voting system or distribute it to other county Democratic organizations for use despite Otto's feedback. Consequently Otto feels ethically bound to share this document with the public before the voting system or a derivative system is used again.*

Review and Partial Audit of A Voting System

EXECUTIVE SUMMARY:

Otto Smith was asked by David Griffith, a candidate, to sit in as an observer at the LD24 district election on Jan 10, 2021. The election was implemented with an online remote voting system developed by the Jefferson County Democrats (JCD). Smith determined that the user interface for the voting system contained several security problems, making it vulnerable to both external hackers and internal malfeasance. Although Smith does not believe that hacking or internal malfeasance occurred, he strongly believes that solutions should be pursued to remedy the system's vulnerability.

Smith presents 3 possible solutions to remedy the problems inherent in the current system and offers his opinion that using existing free and open-source code, such as Helios, would be the best solution.

INTRODUCTION:

David Griffith, a candidate, asked me to sit in as an observer at the LD24 district elections on Jan 10, 2021 to insure and witness to accuracy and fairness. The election was implemented with an online remote voting system developed by the Jefferson County Democrats, herein the JCD system. The election ran smoothly. The user interface was slick and it is clear many hours of volunteer time had been spent in developing the voting system. On the surface and to users, it appeared to work very well. Unfortunately, it contained several security problems that would allow it to be hacked. In many cases such a breach would be difficult to detect. I was able to discover these problems with the help of Libby Urner and Bruce Cowan who walked me through how the vote was carried out and gave me access to the code to audit in preparation for the meeting.

I have no reason to believe that the vote on Jan 10, 2021 or a previous vote on Dec 13 2020, was either hacked or manipulated, although in both cases detection would be difficult. If I were asked if I had confidence in the vote, I would say "I do not have confidence" but I believe that all players were acting with integrity.

In this report I am talking specifically about the JCD system, and when I talk about other systems, it is only about secret ballots in which the user (Alice or Bob) expects the following:

1. Their vote will be counted and counted correctly.
2. Their identity can not be associated with their vote by anyone. That includes those operating the election system as well as others participating in the vote.
3. That only people authorized to vote will be allowed to vote.

DEFECTS IN THE SYSTEM

A. Open to external hackers:

I will start with the most serious defect found. The system was built on top of Google Forms and Google Sheets. Both these programs depend heavily upon JavaScript. In particular Google Forms is principally a client side program running on the users machine, the voters machine in this case. For security each voter was sent a "unique_id" a short time before the meeting, and this was the only mechanism used to insure that the voter was identified to be able to vote and was also the mechanism used to find the voter if there were problems. The complete list of all of the secret ids was stored in a giant regular expression designed to match any unique_id when entered in the google form. Since this is client-side code, anyone (Chuck or Craig) who had access to the voting web address during the period of the election also had access to everyone else's unique_ids. The only other info collected on this page was an email address. The email address was used to send a receipt to the voter and was not checked against any list of approved emails. Any email could be entered. The only security was the unique-id.

A vote is taken as follows: Alice has in their possession a unique_id which is a combination of credential and password that was mailed to them a day before the meeting. Immediately before a vote, Alice is given the address of a website. When Alice accesses the website, they are presented with a page that collects two pieces of information, an email address and the unique_id. The unique_id is verified and Alice is allowed to vote on the next page. When the vote is complete, Alice is sent an email confirmation containing their vote.

For Chuck, an authorized voter, to hack the system, (we assume they are using Firefox):

1. Chuck goes to the website to vote.
2. Chuck goes to the menu item:
   Tools -> Web Developer -> Page Source
3. Chuck copies the page (Right Click -> Select All --then--- Right Click -> Copy)
4. Chuck pastes the source page into a suitable editor.
5. Chuck searches the page for their unique_id call it <chuck-id>. They will find a string that looks something like the following:
   ... <id>|<chuck-id>|<id>|<id> ...

This is a complete list of all user id's separated by the "|" character. Craig, who is not authorized to vote and has no unique_id, can access the web site during the vote, they can do the procedure outlined above, and can search using a regular expression and find the full list of unique_ids as well. At this point Chuck or Craig can vote any one of these ID's as well as copy them for later nefarious purposes. Assuming Craig is bent on destruction, Craig could alter the JavaScript code on their own machine to overwhelm the system either in a denial of service or a vandalism attack.

Even if the user ids were stored on the server alone, storing them in plain text rather than hashed is a well known security problem. Essentially all logins on all sites require a user defined PASSWORD and do not store the PASSWORD on the system but store a hash of the PASSWORD. That is why the operators of such sites require a PASSWORD reset when Alice or Bob have forgotten their PASSWORDs, since there is no way for an administrator to recover a PASSWORD for Alice or Bob.

I was told that for security or Robert Rules reasons, I am not sure which, Alice and Bob must be allowed to change their vote by voting more than once and that only the last vote would be recorded. This is not security nor can I find it in Roberts Rules. This allows a Chuck or a Craig who has Alice's unique-id to be able to hijack their vote. Chuck or Craig can use any email address for the receipt so Alice is not able to verify their vote with a receipt. In the JCD system, some less common mail-readers fail to display the receipts correctly at all. I asked several times how the voter-ids were generated, but this part of the system has not been disclosed to me. Unique_ids are short, in the Jan 10 election four capital characters long. There are less than 400,000 possible

unique_ids. Another way to attack the system is through a simple brute force attack. This can be fast since the attack is on the users machine and can be done with a malicious JavaScript program.

B. Open to internal malfeasance:

How the whole of the JCD vote was carried out behind the scenes, "under the hood" as Bruce Cowan has described it, is still unclear to me. I have not been shown any written document describing and naming the roles and jobs in detail and the few that have been described to me are all vulnerable to cheating. I want to be clear here that I do not think any cheating took place in the Jan10 vote. I believe everyone acted honorably, but a system in which the procedures are so lax as to allow multiple points for malfeasance is highly undesirable. The person answering a phone and voting for a member who has trouble using the web interface has access to this person's telephone number. I will call this person Telly. When Bob gives their vote to Telly, Telly can identify them by their voice, by their telephone number, and by their return email address. Telly votes for Bob. Telly knows everything needed for Bob to vote including their unique_id and their vote. Nor can Telly forget all that they hear. Telly will know, willingly or unwillingly, how some people voted. In the JCD system, Telly is a person Alice and Bob are forced to trust and who in most cases is unknown to them. I believe Telly enters votes with exactly the same interface as every other voter. They could in fact enter the vote wrong, or enter a vote wrong with a different email address at a different time. This would be cheating and requires no sophistication to implement. If Bob can not vote from his own machine, Bob would be better off to call his good friend Alice on the phone and ask them to make the vote. For some reason this was not considered a permissible alternative for voters in general, but only in special cases. Telly should not be part of the system at all. I think Telly was included in the system because the designers of the JCD system, who also operate the system never realized that they are the folks who should be distrusted more than other voters. In a Democratic voting system, the system needs to be designed so the system is trusted, not the operators of the system. The system should be designed so that operators of the system are constrained to act like machines.

"Martin" refers to the person who manages the spreadsheet to which the vote data from google forms is sent. Martin is tasked with both monitoring the votes as they come in and after the vote is closed, eliminating all but the last of duplicate votes, and performing the final tally. This spreadsheet is never shown to Alice or Bob, only the result of the tally is shown. A person, Oscar herein, is assigned to watch in real time the same spreadsheet in google sheets where the vote data is captured to make sure that Martin manipulates and records the data correctly. Martin could with very little effort capture data on one sheet, manipulate it, and send it to a second sheet which is the sheet that Oscar is actually watching. There is no check to insure that any of the code and system that Martin is using is correct, running where it is supposed to run, and has not been written with a malicious intent. The spreadsheet on which the data is collected contains a hidden row with all of the voters emails and a visible row with the unique_ids. Martin could easily peek at the hidden row for a number of reasons, possibly surreptitiously in order to discover a voter's real identity. In addition, since the unique-ids are displayed to Oscar as well as Martin, either Martin or Oscar could easily vote one of the unique_ids without the other or anyone else knowing. Given the number of possible leaks of unique_ids, if a breach were detected, it would be difficult if not impossible to locate the breach. Was it Oscar, Martin, Telly or a Chuck or Craig that breached the system?

Everyone who runs the JCD system including Oscar, Martin, Telly and others including a person I believe who has a list of names and their associated unique ids whose role was mentioned by Bruce Cowan, but not described in detail, is capable of cheating and/or disrupting the vote. In addition, some of these people also have the means to discover how individuals voted and those who can not discover this by themselves can conspire with others to associate a vote with a voter. There are undoubtedly, given this number of security problems, more ways to breach the system.

SOLUTIONS FOR REMOTE VOTING

A. Rationale

Let's step back and look for some solutions. Some people think the end of Covid will allow us to go back to old ways, but we don't know when Covid will end, we don't know when and if a new pandemic will occur, and there are many other reasons why people may not be able to gather as casually as they used to in this new and heating world. For me at least, attending these meetings virtually is less stressful and allows more time for other activities. Virtual meetings could increase attendance. We can save money and time by running more meetings virtually. We need systems such as the JCD system. And we need other automated systems to help with order at virtual meetings so that a modified version of Roberts Rules can be adopted and used successfully. The JCD system is important in the long run as a use-case regardless of whether it worked as designed or not. I am used to systems not working. That is how they are developed. A good system is designed and improved through multiple failures. The JCD system needs to either be fixed, or we need to understand it as a lesson learned and step back and look for other solutions for remote voting for the future. Audits of existing systems are critical since in the future, the ideal would be for the state to approve and advance a single system for all elections in the Washington State Democratic party.

The JCD system developed in house had to meet some very specific requirements that were passed down by the state or are in the bylaws. Various existing systems may not be easily adapted to requirements that:

1. We must elect two people of different Genders.

Required by the bylaws and a good overall provision to promote diversity.

2. Every one should be able to vote more than once.

This is a requirement that reduces robustness. Unheard of in almost all other systems and probably introduced to insure a person could change their vote if a system receipt was wrong. This introduces new problems to cover up an old problem that still exists.

3. Every voter needs to be emailed a vote receipt. Vote receipts are a good idea, but the easily forgeable certificates provided by the JCD system, particularly combined with number 2 above, makes them nearly useless for forensics.

These three requirements were met by the JCD system with considerable work and thought on the part of the designers, but security issues in the whole of the JCD system remain.

B. Specific Solutions

Here are three kinds of possible solutions to the voting problems covered here. They are: Use existing free and open-source code, trust only one agent, and correct the JCD system.

<u>FIRST SOLUTION - Use existing free and open-source code</u>

Two such systems are mentioned here:

https://vote.heliosvoting.org/

More than 2,000,000 votes have been cast using Helios. Super secure free system with all code on git-hub. Lawrence Lessing is an advisor.

https://democraciaos.org

From Argentina progressives. Widely used in Argentina. Probably not for us, but worth looking into.

Implementation issues: Helios, at least, should be easy to install by anyone used to working with command line instructions and who knows how to download programs from git-hub. To understand the installation and more details of how it works, it is best if they understand both Python and JavaScript which are widely used in the system. The Democratic party is heavily dependent upon volunteers and we would need to find someone both capable and willing to install it on a server and test it. After installation, the front end could be learned by a larger number of people. Installation of existing working code is probably less work than building a system from scratch, as the JCD has done, but requires a different kind of computer knowledge. If a person is not comfortable working from the command line and using command line tools to download and install programs, they will not succeed at installation.

The advantages of using this is that it really is secure, or at least its vulnerabilities are minimal since all traffic is encrypted. Modern cryptographic techniques are used to insure both secrecy and accuracy. The cost of breaching such a system becomes so high, that the probability of a breach, particularly in an election that doesn't involve massive amounts of money or power, is small.

The principal problem with these system would be acceptance, since verifying the techniques used requires a degree of mathematical sophistication. On the other hand, Alice and Bob were willing to accept the JCD voting system which was riddled with security problems. We should encourage those who might suspect the code from one of these larger, older and more open systems, to look at the code, if they know how to do that, and/or to look at documentation for the system and/or to investigate the organization that created the code to see if they trust the organizational structure. In an implemented system, we can give read only capabilities to anyone who wanted to browse the code, or wished to compare the code to its source on git-hub. Keeping all of the code and all of the processes open increases security. Furthermore, this FIRST SOLUTION, to use existing free and open-source code, if done under the auspices of the state rather than local Democrats, could produce a usable system residing on a state machine, authorized for use by any party division including county parties. The computational cost would be minimal, would eventually require fewer personal resources to run, and would be open to verification by a larger body of individuals. The Helios system is built by a small open organization. It is a giant mistake to believe that a large company, motivated by money, is more capable of building a system than a non-profit, dedicated to democracy and working on open source and free systems. What we would like to have is the Linux of voting systems. Linux is the most widely installed operating systems in the world. Linux and its derivatives, are open source and in most cases free. Linux runs on the top five hundred fastest super computers in the world. Versions of Linux are used in most routers on home networks, and it is a good bet, that any smart device you have in your house is running a scaled down version of Linux. Linux is the basis for the android operating system that works on most smart phones. The chrome OS used on chromebooks is a Linux based operating system designed by Google. Linux is chosen for these applications because it is free and open source, more robust and reliable and less prone to being hacked than commercial or proprietary systems. Rather than having a secretive crew of developers, it has every computer science student, both undergraduate and graduate level, and hackers and crackers and high school students all over the world testing it, poking at it and improving it. The best choice of a voting system would be the Linux of voting systems, free, open source and widely adopted. Currently there is no leader but there are many free and open-source systems. I have mentioned above only two because they are well respected and I know a little about them. There are many others.

In such a voting system, like Helios above, we are trusting that the code was not written with malicious intent, that the code has been tested and verified, and that the cryptographic techniques are mathematically correct. It may be easier to trust Bob, our next door neighbor spreadsheet whiz, to do a good job of writing a system. We know them personally. We know they are honest, but they may be working beyond their level of expertise and not understand all the implications of a secure system and the need to have it reviewed, audited and tested by a large number of individuals before it is deployed.

SECOND SOLUTION-Trust only one agent

Consider the JCD system above. In order for it to work as designed we must trust everyone who has anything to do with the back end, all those officials who run or observe the election. In fact a large committee of individuals who most voters do not know personally and may not trust at all. Instead of trusting this committee, any one of whom could easily cheat inadvertently, individually, or in collaboration with others, we can design an election with a single trusted human agent. To do this we could pull a Diogenes and find a single honest man, or barring that, we could find a semi-honest lawyer (herein Dana) and bind them with oaths, contracts, and coin, to carry out their functions and keep records as required. Each voter is allowed to contact Dana one at a time, including Alice. Think breakout room, or telephone, or text if necessary. Dana first verifies Alice's credentials, then accepts Alice's vote. Dana enters Alice's vote on a spread sheet and gives Alice back a receipt. The receipt is a unique-random-number. The unique-random-number is entered next to Alice's vote. When everyone has voted, the spread sheet of votes and unique-random-number is displayed for all to see. Every voter can verify that their own vote is correct by looking at the spreadsheet and finding it next to their unique-random-number. Dana is a trusted agent, (bound by oath, contract and coin), and Alice must trust Dana not to divulge any voters unique-random-numbers. But the work of Dana is to a large degree also verifiable by Alice. Alice can count the number of people who have voted and verify that there are not too many, and Alice can verify their own vote was entered correctly. What Alice can not do, and needs to trust, is that Dana has not shared their identity and vote with Bob or any other voter. On the other hand, Dana is bound with oath, contract and coin not to do that. In addition Dana is bound to keep records of everything they did for a period of time and then to destroy the whole lot. In our District I think a large majority of the voters would rationally accept that using Dana is a more reliable and easier to trust system than the current system of the JCD. We can also provide Dana with code and computer systems so that they can do their job more efficiently, but if necessary, given a small election, (fewer than 100 people?) the whole process could be done by hand, but could be vastly accelerated with some automation for Dana.

THIRD SOLUTION - Correct the JCD system

Addressed in fairly great detail above was vulnerability of the JCD system to Chuck and Craig who had malicious intent. It also addressed the need for trust in a large number of people. What it did not address is the issue of how voters are credentialed with a unique_id initially and how they secure their credentials. I have not reviewed how this is done in any open-source and free systems, even the two mentioned above, so this portion of this review is more off-the-cuff. With that caveat, the JCD vote administrators did not answer me when I requested information on how the secure_ids were generated and assigned, nor was I ever given a written document explaining how the system works. On the other hand, Libby Urner in particular, was very open with granting me access to the google forms and google code she had used in developing her part of the system. This saved me considerable time and effort in finding potential problems.

If not providing information on unique_id generation is an attempt at security, it is seriously misplaced. It is security by obscurity. There is no security in obscurity. My attempt to audit the system was done completely within the constraints granted by the JCD. Like all professionals, there are things we may know how to do that we don't do, and organizations such as the Union of Concerned Scientists have published articles and explanations. There are papers attempting to define ethical hacking vs unethical hacking. I decided, since I had requested an audit, and since I was being granted access to a portion of the actual system, to stay within the confines of what was requested of me. There is clearly a problem with the set of unique-ids which are a combined password and credentials and that are really only secret to the out-group as long as they don't try to crack the system. The availability of this information to the in-group makes every one of them a potential suspect in voter fraud. A considerably simpler and more secure system was the mail in system used in Pennsylvania for the presidential election. Part of that system was video monitors covering every part of the facility where the votes were counted. This surveillance and transparency guaranteed the correctness of the vote. There is no similar surveillance in the JCD system, and because of its massive dependency on trust, even

allowing observers such as Oscar, expose the system to less security rather than more. Oscar becomes a potential security risk.

I would probably have discovered the client-side exposure of unique_ids earlier if I had captured the JavaScript code during my own votes at the JCD elections on Dec 13, 2020. I did not do this, although I admit temptation. Capturing this code would be considered ethical by many since a computer owner should be able to audit code put on their machine through other websites, in order to protect themselves against malicious attacks, but I chose not to do this. Once given access to the code in my role as an observer, it was easy to detect the particular problem of everyone's unique-ids being exposed in the JavaScript code running on the users machine. A password that is granted by someone else, in this case the unique_id, was suspect from the start. Because of other errors in the system, I am fairly sure that the unique_ids are not generated in a manner that makes it impossible for members of the in-group to associate names with unique_ids and that the generation and mailing of these unique_ids is not automated by a trusted program. Regardless, we know that these unique_ids are not secure. But they can be made more secure. All of us have access to many sites where we register and create our own PASSWORD. This type of system should be a wrapper for any online voting system. I offer here a little detail about how this works "under the hood," to use Bruce Cowan's phrase. As soon as Alice receives their unique ID they need to secure it with a loginPASSWORD they invent. Systems to do this are widely available and used in almost every login site on the web. The loginPASSWORD is concatenated with some other info called sugar and the unique_id, and is hashed with a cryptographic hash such as MD5, SHA1 or preferably SHA-256. The hash value is stored in a table with the unique_id. For security, the loginPASSWORD is not stored. It is forgotten by the system and Alice is the only person who knows their loginPassword. When Alice logs into the system again to vote, they provide their unique_id and their loginPASSWORD and the hash is performed again. If the hash values match, then Alice is allowed to continue, otherwise an error is generated. It would be simple to take an already developed login system and site and embed the rest of the voting code in this wrapper to verify that only authorized voters like Alice and Bob can get into the system.

A second hardening step would be on the client side, to allow Alice and Bob to generate a verifiable vote. Again this can be achieved with a simple SHA-256 hash. The JavaScript program on Alice's computer collects a vote and a votePASSWORD. The votePASSWORD can be anything Alice wants. The JavaScript program generates a voteHASH from the votePASSWORD, their unique_id and their vote. This serves the same function as the unique-random-number in SECOND SOLUTION, Trust only one agent, above. Alice's vote is the pair of the voteHASH and their actual vote. The forms program displays this vote to Alice including Alice's voteHASH and gives Alice the opportunity to mail or text it to themself. Alternately Alice can take a screenshot or download it, or they can just write it down. The system accepts this vote because it knows Alice logged into the system correctly. The product of this system after the vote is a spreadsheet in which Alice's voteHASH and her vote are recorded along with Bob's and everyone else's. The spreadsheet is open and viewable by all. Every one can then find their own voteHASH and verify the vote was correct. And in fact if the vote next to Alice's voteHASH is wrong, Alice can prove it is wrong by disclosing their votePASSWORD. Only they can generate the voteHASH using their votePASSWORD and their real vote. No one else can do this. And unless shared, no one knows anyone else's voteHASH so they can't determine from the public spreadsheet with everyone's vote on it how anyone else voted. But they can verify their own vote is entered correctly.

This system depends upon a cryptographic hash such as SHA-256

Properties of SHA-256

1. The algorithm is well defined and well known and easily implemented.
2. It is available as a library in most systems.
3. Where it is not available as a library it may be available as free code.
4. Implementations are available on the web:
   See https://xorbin.com/tools/sha256-hash-calculator
5. A result of a hash will look something like this when printed in hex:
   5feceb66ffc86f38d952786c6d696c79c2dbc239dd4e91b46729d73a27fb57e9

And most importantly, nobody has ever found any two documents that have the same SHA-256 hash and given a

SHA-256 hash, nobody has ever been able to find the document that produced it from the Hash alone without just trying hashes of suspected document strings. To try them all, (the number of possibilities is around 10 raised to the 77th power, which is a good estimate of the number of all of the elementary particles in the universe) would literally take an astronomical amount of time.

This option is a moderately secure way of providing a voting system that could be built moderately rapidly with current tools and probably could use a portion of the code that already exists in the JCD system. There are a number of problems that could occur that are beyond the scope of this document which may already have failed on the too-much-information grounds, but which we could discuss in another forum.

DISCUSSION OF OPTIONS ON WHERE WE GO FROM HERE:

Option 1. Forget the whole thing, kick it down the road. I suspect this is the most likely outcome. I would argue against this on the basis that we will probably need a remote voting system in the future, either when this pandemic fails to end as quickly as we hope, or a new pandemic happens, or we can no longer afford to drive any cars, particularly ones that run on gasoline, or that more than half of our members are confined to their homes for a number of reasons. In the long run, kick-it-down-the-road is the most expensive way to deal with these problems.

Option 2. Try out a free open source voting system. Helios would be a good first choice. Install it and run it, if need be change it, and submit changes back through git-hub to its authors for approval. We would be participating in a larger effort, in fact, if you will tolerate the word socialist; a socialist effort in which we cooperate with a larger society and play our limited part. We will contribute to this effort simply by using the system as beta testers. But we can do more by improving a widely adopted system provided free and as open source built by generous good intentioned people with more knowledge about how secure systems work than we have locally, certainly more knowledge and experience than I have. We could also help with documentation and writing up our user experiences. This is my first choice of options.

Option 3. We rewrite and harden the JCD system. This is a better choice than option 1. In fact, although the authors of the JCD system may be suffering anxiety and stress due to the exposure of its failings, like all "failed" experiments, it was not a failure in the sense that we have learned from it and others can learn from it as well. The authors behaved ethically in opening up some of the source to examination, although if we continue down this path, all of the code and process should be completely open. I would suggest a copyright by the authors by either a creative commons license or a license to any Democratic Party. A fundamental requirement for a secure system is that the source code be open and we should never ever again use any system where this is not true. A general failing that I did not mention anywhere else is that Google Forms and Google Sheets are proprietary code, although free to use by anyone. In reality, proprietary code should only be used if we can verify all traffic through those systems is secure and validated. This may not be easy, and in general we should not use proprietary code for any secure portions of the system. To a degree, we can blame the proprietary nature of google forms to the failing of the unique_id system. The google code is long and complex and portions of it are hidden from users, it is not surprising that a writer of a front end system with no computer science or security training, was unaware of these problem and the differences between security client side versus server side.

Option 4. Trust in a single person bound by Oath, Contract and Coin. This was a suggestion offered only partly in jest. It is a little half baked, but really should be considered and maybe developed further seriously. What is it we are willing to trust? Will we trust computer code we haven't seen and haven't audited more than an individual? More code and more layers of security do not necessarily increase the robustness of a system. More code and more layers are far more likely to increase features and functions at the expense of robustness and reliability. This argues for a system that has minimal code in it and is, in addition, completely open, such as the paper ballot systems used in small meetings. So possibly any computer system should emulate this as close as possible which is what the trust in a single person tries to do.

CONCLUSION

My belief is the best system would be a system that is designed to be mathematically correct and which some of us can at least partially verify. A system which we test before we use, and where we depend upon others as well as ourselves to stretch it, poke at it, and attempt to break it. Quality programs may be initially written by individuals, but they are made into full usable systems through the cooperative work of many players. Option 2 above advocates for adopting the first solution, to use existing free and open-source code. This is probably the best approach and I would be happy to help on installation and testing. I am not ruling out help on Option 3 or 4, but these would need written design documents.

Best Regards To Everyone

Respectfully Submitted,
Otto Smith