Detecting Malicious Poll Site Voting Clients

C. Andrew Neff*

September 15, 2003

Abstract

We describe a cryptographic approach to ensuring the end-to-end integrity of elections which use digital (electronic) ballots. Security properties are based entirely on the information available to election participants, and thus do not require one to assume that the machines responsible for collecting votes are trustworthy. The end result is a large scale election system with well understood, openly accessible audit and dispute resolution properties – characteristics that have typically belonged only to small democratic processes such as a "show of hands." However it also maintains a secret ballot, and protects against vote selling or vote coercion.

1 Introduction

Large scale elections have always been difficult to protect. As a democratic process, every election participant has a legitimate interest in assuring the integrity of the final count, yet size and scope have, in recent decades, dictated that access to the counting and audit process be significantly limited. This *"minority count"* scenario increases the chances that a small group with selfish interests may un-detectably change the election outcome.

With the current use of electronic election equipment (DREs) the situation has only been worsened, since it is much easier for individuals or small groups with access to the machines to successfully and un-detectably take

*aneff@votehere.net

subversive action. The trouble is that old processes used to protect the integrity of paper ballots are ineffective in the digital domain.

A more robust system can be built by leveraging the strengths of digital information rather than trying to imbue it with the properties of paper. Because modern communications infrastructure can support essentially unlimited access to information, digital ballot data can potentially be made accessible to all election participants, even in the case of very large scale elections. Such an *"open count"* system is inherently better protected against fraud than a minority count system – paper or electronic – regardless of the certification processes it has been through.

For an open count system to work however, two critical properties must be maintained. First, ballot secrecy must be protected – even for voters faced with the threat of coercion. And second, any fraud or election data inconsistency must be provable beyond dispute. (It does little good to have a system where election participants can see fraud, but are not empowered to correct it.) The purpose of this paper is to describe a method for maintaining both of these properties in an open count system using digital data. With it, an election is protected with a high degree of certainty against undetected fraud by any party, *including* the poll site voting devices themselves.

1.1 Assumptions

1.1.1 Ballot Format

For most of this discussion, we will assume that the ballot in question consists of a sequence of $\mathcal{Q}$ "questions" (these might also be called "issues" or "candidates"), and that for each question, Q, voters are allowed to choose an ordered sequence of m_Q "responses" from a list of $n_Q \geq m_Q$ fixed "answers". We call ballots that have these properties *standard ballots*. Standard ballots support most types of election tabulation methods. For example, preferential voting can be supported by sequencing voter responses to a particular question in order of the voter's preference. In fact, to simplify notation, we will only discuss the specifics of the case $\mathcal{Q} = 1$ and $m = m_Q = 1$. There are no special difficulties that occur in the case of larger values. The reader will easily see that the methodology allows more complicated ballots to be viewed as a collection of "parallel elections" with $\mathcal{Q} = 1$ and $m = 1$.

Standard ballots do not include “write-in” responses however. At the end of this paper, we will discuss how “write-in” responses can be supported, but they will not be protected by all of the security properties that protect standard ballots. This fact is not a limitation of the approach we take, but rather due to a coercion problem inherent to write-in responses. In short, a vote coercer can determine if a voter voted the entire ballot according to instruction by demanding that voter provide a known unique string as a particular write-in response. (This exploit has actually been used by corrupt regimes.)

1.1.2 Voters Indistinguishable

We also assume that the vote client is incapable of distinguishing voter identities. In particular, the client can not predict whether a given voter is more or less likely to “check its behavior.” This is an important assumption, but it can be realized in practice through a combination of vote process design, and restrictions on machine hardware. Consequently, we also assume that BallotSequenceNumbers are unpredictably assigned to voters.

1.2 Notation

The appendix contains a Glossary of Data Structures which is not limited to the case $\mathcal{Q} = 1$, $m_Q = 1$. It will be useful to refer to it in the analysis that follows. The reader should keep in mind, however, the $\mathcal{Q} = 1$, $m_Q = 1$ simplification we have adopted for the sake of discussion.

We call special attention to the CryptoElectionParameters structure which contains notation for the cryptographic quantities that will referenced throughout this paper. The data in the CryptoElectionParameters is determined, and published, well in advance of the election so that its contents can not be disputed.

2 Vote Receipts

Already, under the procedures currently used at poll sites, open publication of voted ballot data is not harmful. This is because the link between voter (individual) and ballot is procedurally lost at the poll site. This is true even if unique identifying marks, or *ballot sequence numbers* (BallotSequenceNumbers), are assigned to each ballot, as long as the procedure by which each voter obtains a BallotSequenceNumberis sufficiently unpredictable. In the paper ballot setting, one can imagine each voter choosing a blank ballot paper from an arbitrary place in a large pile of blank ballots, and procedurally destroying the ballot papers that are left at the end of the voting day. There are better ways to achieve arbitrary distribution of BallotSequenceNumbers with electronic ballots, but we will not discuss them in detail here.

Assuming then that voters know their own BallotSequenceNumber, but that no other election participant does, publication of voted ballot data allows all voters the chance to verify that their ballot was correctly included in the final count. Any misbehavior by the poll site voting machine would be detected by the process of verification, *however* this is of limited value since voters have no way of proving that their ballot was altered. Since it is unreasonable to assume that all voters are honest, election integrity can not be firmly established. Further, the very act of mounting a protest spoils the voter's right to ballot secrecy.

Financial transactions have long dealt with this issue by the mechanism of a receipt. By issuing a "vote receipt" of indisputable authenticity to voters after they have confirmed and committed their choices, the problem of election dispute can be resolved. However, the act of protest would still spoil the voter's ballot secrecy, and a new, more sinister problem is created: Since voters have been enabled to *prove how they voted*, they are vulnerable to the threat of vote coercion. What is required is an indisputable receipt, for which only the voter can determine a meaningful connection to specific ballot responses.

3 Voted Ballot Representation

The specific form of a receipt with the properties we seek will unavoidably be determined by the accepted representation standard for blank and voted ballots. For our discussion, a BlankBallot is simply a BallotSequenceNumber along with an ordered sequence of AnswerMarks. The order of the AnswerMarks determines precisely how they correspond to the available ballot answers. A VotedBallot is simply a BallotSequenceNumber along with a single VotedAnswer (ElGamalPair).

4 Ballot Codebooks and Commitments

Definition 1 A VoteVerificationCodebook, $\mathcal{C}$, is a map from AnswerMarks to character strings.

We will only consider a special subset of VoteVerificationCodebooks, namely the parameterized family of maps

$$\mathcal{C}_\alpha(A) \;=\; H(\gamma_A^\alpha) \tag{1}$$

where H is a publicly known "shortening function". (One can think of H as simply truncation to a fixed length.) For practical reasons, H must also have the property that the probability over α of $H(\gamma_A^\alpha) = H(\gamma_B^\alpha)$ for $A \neq B$ is small.[1] This property can be assured for any reasonable H simply by making the length of its output sufficiently long. (20 bits should be more than sufficient.)

Definition 2 A CodebookCommitment, C, is an element of the ElectionEncodingSubgroup.

Clearly there is a unique, publicly verifiable correspondence between CodebookCommitments and VoteVerificationCodebooks, namely

$$C \;\longleftrightarrow\; \mathcal{C}_{\log_g C} \tag{2}$$

[1] A variation on the method presented here eliminates the need for this restriction.

However, $\alpha = \log_g C$ is protected cryptographically even if C is known. This affords the possibility of creating CodebookCommitments through a multi-authority *secret sharing* process. If

$$C = C_1 \ldots C_n \tag{3}$$

and each $C_i = g^{\alpha_i}$ is constructed by a separate VoteVerificationTrustee, then

$$\alpha \doteq \log_g C = \sum_{i=1}^{n} \log_g C_i \tag{4}$$

and α is kept secret unless all n VoteVerificationTrustees share their secret α_i.

5 The Election Protocol

The open count election methodology we propose consists of the following steps:

Election Preparation

EP. 1. Shared election cryptographic parameters are created as described in [5].

EP. 2. The n VoteVerificationTrustees agree on a collection of N_V BallotSequenceNumbers, $\{b_i\}_{i=1}^{N_V}$. (N_V must be larger than the number of eligible voters.)

EP. 3. Each of the VoteVerificationTrustees, T_j, prepares a fixed sequence of CodebookCommitments, $\{C(i,j)\}$, where $\alpha(i,j) = \log_g C(i,j)$ is randomly generated by T_j and kept secret. We let

$$C(i) \doteq \prod_{j=1}^{n} C(i,j)$$

(See TrusteeCodebookCommitments.)

EP. 4. The entire collection of $C(i,j)$ is signed and published.

Voting

Each voter executes the following steps

V. 1. The vote client *commits* a readable representation, D_0, of $\mathcal{C}(i)$. Failure to do so is immediately detectable by voter, so we henceforth disregard this condition.

- For example, this can be done by paper printout, but other options are available.
- If, by chance, $\mathcal{C}(A) = \mathcal{C}(B)$ for some $A \neq B$, the voter may demand a new BallotSequenceNumber. (By choice of H, this happens with negligible frequency.)
- Procedurally, the voter should be prevented from leaving the poll booth with D.

V. 2. Voter selects a response, R, from the available answers.

V. 3. The vote device provides a *signed* copy of $\mathcal{C}(R)$ (the vote receipt, or VoteVerificationStatement) to the voter. Failure to do so is immediately detectable by voter, so we henceforth disregard this condition.

V. 4. The vote device records the VotedBallot corresponding to γ_R. In our simplified election model, this is essentially the ElGamal pair (X, Y) where $X = g^\sigma$, $Y = h^\sigma \gamma_R$ and $\sigma \in \mathbf{Z}_q$ is randomly chosen.[2] In addition, the vote device must also attach a *validity proof* demonstrating that this is an encryption of *at least* one of the possible γ_A. (Examples of such proofs can be found in [4] and [5].)

Tabulation

T. 1. The entire list of VotedBallots is published with associated validity proof created by the vote device(s).

T. 2. For *each posted* VotedBallot, the VoteVerificationTrustees cooperate to verifiably compute, and publish, a CodebookVerificationCode, D_1. This is accomplished by

1. VoteVerificationTrustee j computes

$$(X_j \, , \, Y_j) \; = \; (X^{\alpha_i} \, , \, Y^{\alpha_i}) \tag{5}$$

[2]For coercion resistance, it is important that σ be chosen randomly and kept secret. Currently, this must be done with proper procedures for managing secret data. In a future version of this paper we will discuss an extension to the voting protocol that protects against this threat without relying on procedure.

2. VoteVerificationTrustee j publishes (X_j, Y_j) with a corresponding pair of Chaum-Pedersen proofs ([6]) demonstrating that equation 5 holds.

3. The VoteVerificationTrustees cooperate (see [11]) to decrypt the pair

$$(\bar{X}, \bar{Y}) \doteq (\prod_{j=1}^{n} X_j \, , \, \prod_{j=1}^{n} Y_j) \tag{6}$$

They publish the decryption, γ, along with decryption validity proofs exactly as in [5]. (From this anyone can derive $D_1 = H(\gamma)$. That is, D_1 is effectively published by publishing γ.)

T. 3. The VoteVerificationTrustees tabulate the entire set of VotedBallots by way of a *verifiable mix (shuffle)*, and publish the entire mix transcript. (See [9] and [10].)

Voter Verification

Voters protect the contribution of their response (ballot choice) by

C. 1. Immediately checking the receipt signature to be sure that they leave the poll site with indisputable evidence of their CodebookVerificationCode, D_0.

C. 2. Check that the published CodebookVerificationCode, D_1 is exactly the same as D_0.

6 Protocol Consequences

Let A be the voter's intended choice, $\bar{A}$ be the choice that is actually encrypted by the vote device, $\mathcal{C}_{PS}$ be the VoteVerificationCodebook displayed to the voter in the poll site, $C = C_1 \cdots C_n$ be the voter's published CodebookCommitment (referenced by BallotSequenceNumber), $\mathcal{C}_C$ the VoteVerificationCodebook corresponding to C via equation 1, D_0, D_1 the two CodebookVerificationCodes described in the protocol, and let δ_T be the influence of the voter's published VotedBallot on the final tally.

- Consistency of the relationship

$$\mathcal{C}_{PS}(A) \longleftrightarrow D_0 \tag{7}$$

  is protected by voter inspection at vote time.

- Consistency (equality) of the relationship

$$D_0 \longleftrightarrow D_1 \tag{8}$$

  is protected by voter inspection of the published tabulation transcript.

- Consistency of the relationship

$$D_1 \longleftrightarrow \mathcal{C}_C(\bar{A}) \tag{9}$$

  is protected by *public inspection* of the the validity proofs in step **T.** 2.

- Consistency of the relationship

$$\bar{A} \longleftrightarrow \delta_T \tag{10}$$

  is protected by the verifiable mix transcript.

If $\mathcal{C}_C = \mathcal{C}_{PS}$, then equations 7- 9 imply $A = \bar{A}$. Equation 10 then implies

$$A \longleftrightarrow \delta_T \tag{11}$$

In words, the voter's intent has been tabulated correctly.

7 Protecting the Poll Site Codebook

The conclusion of the previous section was that preservation of voter intent is ultimately reduced to assuring that

$$\mathcal{C}_C = \mathcal{C}_{PS} \tag{12}$$

To prevent the threat of coercion however, the contents of C_C must be kept secret outside of the voter's poll booth experience.

The dilemma is resolved by employing a cut-and-choose approach. Specifically, we allow voters, or special observers simulating voters, to verify a $\mathcal{C}$

committed by the vote device without using it to vote. In election language, they may "spoil" a ballot in order to check the correctness of its codebook as displayed by the voting device. As long as audits are indistinguishable from actual voters up until the point that the poll site codebook is committed by the device, the accuracy of all poll site VoteVerificationCodebooks can be assured to a high level of confidence. (A complete probabilistic analysis of confidence levels with respect to electorate size and audit frequency is forthcoming.)

Observers may easily check equation 12 for a given BallotSequenceNumber by demanding that the VoteVerificationTrustees all reveal their corresponding secrets. Conversely, the VoteVerificationTrustees can each ensure that no VotedBallot corresponding to an audited BallotSequenceNumber is included in the final tally.

8 Vote Privacy and Coercion Resistance

Extensive research on the *Discrete Logarithm Problem* and *Decision Diffie-Hellman Problem* ([2], [3], [Mau94], [MW98], [Odl85]) indicates that the collection of information contained in the vote receipt *and* election transcript does not give the voters any aid in proving the value encrypted by their published VotedBallot, (X, Y). This is because

- The tabulation transcript is zeroknowledge ([10]).
- Given γ, the VoteVerificationTrustee decryptions are statistical zero-knowledge simulatable.
- Deciding if $\log_{\gamma_A} \gamma = \log_g C$ for any given AnswerMark, γ_A is intractable.

References

[1] J. Benaloh, M. Yung. Distributing the power of a government to enhance the privacy of voters. *ACM Symposium on Principles of Distributed Computing*, pp. 52-62, 1986.

[2] D. Boneh. The Decision Diffie-Hellman Problem. Lecture Notes in Computer Science, vol. 1423, 1998.

[3] D. Boneh, R. Venkatesan. Hardness of Computing the Most Significant Bits of Secret Keys in Diffie-Hellman and Related Schemes. Princeton University Technical report, TR-515-96.

[4] R. Cramer, I. Damgrd, B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. *Advances in Cryptology - CRYPTO '94*, Lecture Notes in Computer Science, pp. 174-187, Springer-Verlag, Berlin, 1994.

[5] R. Cramer, R. Gennaro, B. Schoenmakers. A secure and optimally efficient multi-authority election scheme. *Advances in Cryptology - EUROCRYPT '97*, Lecture Notes in Computer Science, Springer-Verlag, 1997.

[6] D. Chaum and T.P. Pedersen. Wallet databases with observers. Advances in Cryptology - CRYPTO '92, volume 740 of *Lecture Notes in Compute Science*, pages 89-105, Berlin, 1993. Springer-Verlag.

[7] T. ElGamal. A public-key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, IT-31(4):469-472, 1985.

[8] A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone. Handbook of Applied Cryptography, CRC Press, 1997.

[Mau94] U. Maurer. Towards the Equivalence of Breaking the Diffie-Hellman Protocol and Computing Discrete Logarithms. Cryptology - CRYPTO '94, Lecture Notes in Computer Science, pp. 271-281, Springer-Verlag, 1994.

[MW98] U. Maurer and S. Wolf. The Relationship Between Breaking the Diffie-Hellman Protocol and Computing Discrete Logarithms. SIAM Journal on Computing, Vol. 28, No. 5, pp. 1689-1721, 1999.

[9] C.A. Neff, A Verifiable Secret Shuffle and its Application to E-Voting. *Proceedings ACM-CCS 2001*, 116-125, 2001.

[10] C.A. Neff, Verifiable Mixing (Shuffling) of ElGamal Pairs. http:/votehere.net/vhti/documentation/egshuf.pdf.

[Odl85] A. M. Odlyzko, Discrete logarithms in finite fields and their cryptographic significance, Advances in Cryptology - EUROCRYPT '84, Lecture Notes in Computer Science, Springer-Verlag, 1984.

[11] T. Pedersen. A threshold cryptosystem without a trusted party, *Advances in Cryptology - EUROCRYPT '91*, Lecture Notes in Computer Science, pp. 522-526, Springer-Verlag, 1991.

A Glossary of Data Structures

Answer Mark : For the purpose of the cryptographic protocols, each answer (in the AnswerReference sense) must be "randomly" assigned an element of the election's ElectionEncodingSubgroup.

$$\text{AnswerMark} \doteq \gamma_A \in \text{ElectionEncodingSubgroup}$$

The set of AnswerMarks corresponding to a single BallotQuestion must be must be *distinct*. That is, if A_1 and A_2 are both answers to the same BallotQuestion, then $\gamma_{A_1} \neq \gamma_{A_2}$. In order to effectively implement these properties, AnswerMarks are generated publicly as a SHA-1 of Election, Precinct, and AnswerReference data.

Answer Partial Decrypt : A data structure of cryptographic quantities representing the piece of information contributed by a *single* Trustee for a *single* VotedAnswer contained in a *single* RawVotedBallot.

$$\text{AnswerPartialDecrypt} \doteq \left\{ \begin{array}{l} \text{ModularInt } Z \\ \text{ModularInt } c \\ \text{ModularInt } d \end{array} \right\}$$

This represents a valid partial decryption of a VotedAnswer, (X, Y), with respect to an Trustee, A, with KeyShareCommitment, C, if and only if

$$c \;=\; H(g\,,\,C\,,\,X\,,\,Z\,,\,g^dC^c\,,\,X^dZ^c) \tag{13}$$

where H is the system secure hash function (SHA-1).

Answer Reference : A small integer, which, in the context of a fixed election is uniquely associated with a specific (question , answer) pair. Note for the sake of the cryptographic protocols, each ballot question must have its own spcial "ABSTAIN" answer reference.

Answer Text Structure : A specification according to a yet to be determined data standard of the human readable data (display text, title, shorthand name, etc.) associated with a ballot answer (perhaps complicated to support multiple languages, etc.).

Codebook Commitment :

An element of the ElectionEncodingSubgroup. Each CodebookCommitment is used to irrefutably link VoteVerificationCodes to encrypted choices. For proper overall election usage, these must be constructed and *published* prior to voting. For a fixed Precinct, the full set of CodebookCommitments is (multi-dimensionally) indexed by all possible triples: VoteVerificationTrustee, BallotSequenceNumber, QuestionReference.

$$\text{CodebookCommitment} \doteq C \in \text{ElectionEncodingSubgroup}$$

Ballot Answer : This data structure (XML) is needed for both for the blank ballot and for results reporting after tabulation. However, it is only connected to through the interface. We leave specification vague for now, and perhaps even leave the specification to a third party or standards organization. Roughly, it must have an AnswerMark which is *unique for the ballot in question*, an AnswerReference which is also *unique for the ballot in question*, and an AnswerTextStructure.

$$\text{BallotAnswer} \doteq \left\{ \begin{array}{c} \text{AnswerReference} \\ \text{AnswerMark} \\ \text{AnswerTextStructure} \end{array} \right\}$$

Ballot Box Node : The data structure stored in the electronic ballot box as a result of a "successfully cast" ballot. It consists of the ValidatedVotedBallot received, and the PreVerificationCodes which was computed and returned. Of course, only the first element is critical, since the second can be computed from it, but storing the PreVerificationCodes is worthwhile to avoid potential load on the server from a (possibly malicious) voter who is determined to try to cast votes multiple times.

$$\text{BallotBoxNode} \doteq \left\{ \begin{array}{c} \text{ValidatedVotedBallot} \\ \text{VoteReceipt} \end{array} \right\}$$

Ballot Box Partial Decrypt : A vector of BallotPartialDecrypts representing the information contributed by a *single* Trustee for *all* the (properly sequenced) RawVotedBallots contained in a RawBallotBox.

$$\text{BallotBoxPartialDecrypt} \doteq \\ (\text{BallotPartialDecrypt}_1 \, , \, \ldots \, , \, \text{BallotPartialDecrypt}_B)$$

Ballot Goo : Miscellaneous stuff – election name, titles, page and display info, etc., associated with the human readable elements of a ballot.

Ballot Partial Decrypt : A vector of AnswerPartialDecrypts representing the information contributed by a *single* Trustee for *all* the (properly sequenced) VotedAnswers contained in a *single* RawVotedBallot.

$$\text{BallotPartialDecrypt} \doteq \left(\text{AnswerPartialDecrypt}_1, \ldots, \text{AnswerPartialDecrypt}_Q\right)$$

Ballot Question : This data structure (XML) is needed for both the blank ballot and for results reporting after tabulation. The specification is left to the application. Roughly, it must have a question text structure (perhaps complicated to support multiple languages, etc.), and a vector of BallotAnswers. *It must* have one distinguished answer, "ABSTAIN". If it is to allow a write-in response, *it must also* have a second distinguished answer, "WRITE-IN".

$$\text{BallotQuestion} \doteq \left\{ \begin{array}{c} \text{QuestionReference} \\ (\text{BallotAnswer}_1, \ldots, \text{BallotAnswer}_Q) \\ \text{QuestionTextStructure} \end{array} \right\}$$

Ballot Questions : A vector of L BallotQuestions

$$\text{BallotQuestions} \doteq (\text{BallotQuestion}_1, \ldots, \text{BallotQuestion}_L)$$

Ballot Secret : A secret value used to encrypt a ballot. It is an element of $\mathbf{Z}_q$.

Ballot Secrets : A vector of $\nu \geq 0$ secret values (elements of $\mathbf{Z}_q$), used to encrypt a ballot. This information is created by the Vote Client at encryption time, and should be carefully forgotten afterward.

$$\text{BallotSecret} \doteq (\alpha_1, \ldots, \alpha_\nu)$$

Ballot Sequence Number : Ballot identifier used for poll site VoterVerification. The set of BallotSequenceNumbers for a given Election (or Precinct) can be considered equivalent to the set of "potential voters" (including *"provisional voters"*) in the Election (or Precinct).

Ballot Sequence Numbers : A vector of V BallotSequenceNumbers

$$\text{BallotSequenceNumbers} \doteq (\text{BallotSequenceNumber}_1, \ldots, \text{BallotSequenceNumber}_V)$$

BSN Codebook Commitments : A structure that represents all CodebookCommitments for a *fixed* ElectionID, *fixed* PrecinctID, *fixed* VoteVerificationTrustee and *fixed* BallotSequenceNumber. That is, a collection of $l \geq 1$

CodebookCommitments, where l is the number of QuestionReferences in the BlankBallot.

$$\text{BSNCodebookCommitments} \doteq$$

$$\left\{ \begin{array}{c} \text{BallotSequenceNumber} \\ (\text{CodebookCommitment}_1, \ldots, \text{CodebookCommitment}_l) \end{array} \right\}$$

Blank Ballot : This structure needs to link together all the cryptographic and conventional information associated with the Election, Precinct, and set of races, candidates and issues that are to be contested.

$$\text{BlankBallot} \doteq \left\{ \begin{array}{c} \text{ElectionID} \\ \text{PrecinctID} \\ \text{CryptoElectionParameters} \\ \text{BallotQuestions} \\ \text{BallotGoo} \end{array} \right\}$$

Broadcast Value : A (random) element of ElectionEncodingSubgroup. These values are generated as part of Key Sharing, and are an essential component of the Pedersen dealerless secret sharing scheme.

Broadcast Values : An XML string containing the BroadcastValue from each authority.

Certificate : A PKI, typically X.509, certificate.

Check Results : An XML structure containing the results of checking or verification.

Cipher Text : A stream of encrypted bytes. In order to be decrypted, you also need a GeneralPurposePrivateKey, an InitializationVector, and an EncryptedSessionKey.

Clear Text Ballot : A direct representation, in the context of a fixed BlankBallot, of a voted ballot – i.e. set of voter choices. Constructed as a vector of $\nu \geq 0$ AnswerReferences.

$$\text{ClearTextBallot} \doteq (\text{AnswerReference}_1\,, \ldots\,, \text{AnswerReference}_\nu)$$

Clear Text Ballots : An XML structure containing one or more ClearTextBallot.

Codebook Commitment :

An element of the ElectionEncodingSubgroup. Each CodebookCommitment is used to irrefutably link VoteVerificationCodes to encrypted choices. For proper overall election usage, these must be constructed and *published* prior to voting. For a fixed Precinct, the full set of CodebookCommitments is (multi-dimensionally) indexed by all possible triples: VoteVerificationTrustee, BallotSequenceNumber, AnswerReference.

$$\text{CodebookCommitment} \doteq C \in \text{ElectionEncodingSubgroup}$$

Codebook Verification Code : An element of the VoteVerificationCodebook which associates a particular VoteVerificationCode with the appropriate QuestionReference.

Committed Trustee :

$$\text{CommittedTrustee} \doteq \left\{ \begin{array}{c} \text{Trustee} \\ \text{KeyShareCommitment} \end{array} \right\}$$

Committed Trustee Set : A set of CommittedTrustee s

$$\{\text{CommittedTrustee}_1, \text{CommittedTrustee}_2, \ldots, \text{CommittedTrustee}_t\}$$

Crypto Election Parameters : All configuration parameters required for execution of the election cryptographic operations:

$$\text{CryptoElectionParameters} \doteq \left\{ \begin{array}{c} \text{CryptoGroupParameters} \\ \text{CryptoTabulationParameters} \end{array} \right\}$$

Crypto Group Parameters : The set of necessary mathematical parameters that can be generated very early – prior to authority selection and Key Sharing.

$$\text{CryptoGroupParameters} \doteq$$

$$\left\{ \begin{array}{c} \text{ElectionModulus } (p) \\ \text{ElectionSubgroupModulus } (q) \\ \text{ElectionSubgroupGenerator } (\mathrm{g}) \end{array} \right\}$$

Crypto Tabulation Parameters : The set of necessary mathematical parameters that are required before voting can begin (even before a BlankBallot can be completed), but are not known until during or after Key Sharing.

$$\text{CryptoTabulationParameters} \doteq$$

$$\left\{ \begin{array}{c} \text{ElectionPublicKey } (h) \\ \text{SecEncryptionBase } (h_0) \\ \text{CommittedTrusteeSet } (\textit{All } n \textit{ Tabulation Authorities}) \\ \text{TabulationThreshold } (t) \end{array} \right\}$$

Decryption Validity Proof : A zeroknowledge proof of correctness for decryption.

Election : Refers to all voting and tabulation issues within an umbrella jurisdiction, or political unit. Each election has one and only one CryptoElectionParameters structure associated with it. However, an Election can be subdivided into Precincts, which each have their own BlankBallot, possibly, but not necessarily, containing distinct questions and/or issues. Tabulation is performed on a Precinct level. For convenience, *Precinct Results* may be aggregated and published as unified *Election Results* data.

Election Encoding Subgroup : The unique order q subgroup of the ElectionGroup, where q is specified in the CryptoGroupParameters structure contained in the CryptoElectionParameters structure of the BlankBallot.

Election Group : The modular arithmetic group specified by the ElectionModulus (p) parameter of the CryptoGroupParameters structure contained in the CryptoElectionParameters structure of the BlankBallot.

Election ID : A UUID for elections.

Election Modulus : The prime integer (p) which determines the ElectionGroup used for VotedBallot encryption. It is specified in the CryptoGroupParameters structure contained in the CryptoElectionParameters structure of the BlankBallot.

Election Node : The data structure encompassing the (unsigned) data loaded by LoadElection. (It also needs to have pointer information to allow for efficient insertion of ValidatedVotedBallot.)

$$\text{ElectionNode} \doteq \left\{ \begin{array}{c} \text{ElectionID} \\ \text{CryptoElectionParameters} \end{array} \right\}$$

Election Public Key : An element, h, of the ElectionEncodingSubgroup. The corresponding *private key* ($\log_g h$, where g is the ElectionSubgroupGenerator) is a secret cryptographicly shared between a set of "election trustees" (TrusteeSet) via a Pedersen dealerless threshold scheme.

Election Results : A data structure containing all the data necessary to display the election results (tally) in an "official" human readable form.

Most likely an XML structure containing general information such as *Election Name*, *Question Text* and *Answer Text* along with corresponding numerical tabulation results.

Election Subgroup Generator : A fixed element, g, of the ElectionGroup which generates the ElectionEncodingSubgroup. In particular, g must satisfy the following relationship with the ElectionSubgroupModulus, q:

$$|g| = q \tag{14}$$

Election Subgroup Modulus : The prime integer (q) which determines the ElectionEncodingSubgroup used for VotedBallot encryption. In addition to being prime, it must also be related to the ElectionModulus(p) by the relationships:

$$\begin{aligned} p - 1 &= q\,r \\ (q\,,\,r) &= 1 \end{aligned} \tag{15}$$

It is specified in the CryptoGroupParameters structure contained in the CryptoElectionParameters structure of the BlankBallot.

ElGamal Pair : A *pair* of *Modular Integers* (ModularInt).

$$\text{ElGamalPair} \doteq (X, Y)$$

Encrypted Data : A collection of data which can be decrypted, given a suitable GeneralPurposePrivateKey.

Encrypted Session Key : A random byte stream that has been encrypted with a GeneralPurposePublicKey. It is used to encrypt a message with a stream cipher. (The message is not encrypted directly with the GeneralPurposePublicKey because that would be too slow.)

Encryption Private Key : A key which can be used for decryption.

Encryption Public Key : A key which can be used for encryption.

Error Structure : Structure for encoding "unexpected" return conditions.

General Purpose Private Key : A key which can be used for both decryption and signature generation.

General Purpose Public Key : A key which can be used for both encryption and signature validation.

Identification Information : An XML string containing identifying information about the owner/creator of a GeneralPurposePublicKeyor GeneralPurposePrivateKey.

Initialization Vector : A short, pseudo-random byte stream that increases the security of a CipherText.

Key Generation Parameters : The parameters determining the number of Trustees (the KeyShareWidth, n) participating in Key Sharing and the number of these (the TabulationThreshold, t) who must cooperate in order to tabulate.

$$\text{KeyGenParameters} \doteq \left\{ \begin{array}{c} \text{CryptoGroupParameters} \\ \texttt{int}\ 1 \leq n\ (\text{KeyShareWidth}) \\ \texttt{int}\ 1 \leq t \leq n\ (\text{TabulationThreshold}) \end{array} \right\}$$

Key Share Commitment : A modular integer with constraints based on the election crypto parameters

$$\text{KeyShareCommitment} \doteq C \in \text{ElectionEncodingSubgroup}$$

where

$$C = g^s$$

and

$$s = \text{SecretShare}$$

Key Share Width : A positive integer (n) that specifies the total number of Trustees officially participating in Key Sharing.

Keys : A vector of *key* items.

Modular Integer : A BigInt

$$\text{ModularInt} \doteq \left\{ \text{BigInt } x \right\}$$

(Modulus is determined from context, not explicitly represented.)

Multi-Set Element : A data pair

$$\left\{ \begin{array}{ll} \text{ModularInt } \gamma & (\text{an element of ElectionEncodingSubgroup}) \\ \texttt{int}\ \text{count} & \end{array} \right\}$$

Pair-wise Secret : Each Authority evaluates his polynomial at all of the TrusteeEvaluationPoint ID values, including his own.

$$\text{PairwiseSecret} \doteq (ID_i\,,\ ID_j\,,\ f_i(\beta_j))$$

The identifiers (ID_i, ID_j) designate the sender (ID_i) and the recipient (ID_j) authorities. Each ID is the *fingerprint* of the corresponding Trustee object.

Pair-wise Secrets : An XML string containing the PairwiseSecret from/to each authority.

Partially Decrypted Ballot Box : A data structure containing all the decryption information necessary to both *tabulate* and *verify* with respect to a given CryptoElectionParameters (or, with respect to a given SignedBlankBallot structure, which contains a unique CryptoElectionParameters structure). The count is is only verifiable against the contained RawBallotBox component. Additional verification is needed to certify that the count is derived properly from the official set of SignedVotedBallots.

$$\text{PartiallyDecryptedBallotBox} \doteq \left\{ \begin{array}{c} \text{RawBallotBox} \\ \text{TrusteePartialDecrypts} \end{array} \right\}$$

Permutation : An XML structure with attribute Size=n and data containing a random ordering of the numbers 1 through n.

Precinct : Refers to an *"atomic"* sub-jurisdiction, its ballot and tabulation results. "Atomic" means that

1. All voters in a given Precinct must use (i.e. vote on) the same BlankBallot.
2. All ballots cast in a given Precinct must be tabulated together to produce a *single count*, or set of question/issue results. *Seperation of voters within a precinct into sub-categories is not allowed.* If a Precinct needs to be subdivided, it should be seperated into multiple Precincts before ballot casting begins (i.e. polls are opened).

Precinct Codebook Commitments : A structure that represents all CodebookCommitments for a *fixed* ElectionID, *fixed* PrecinctID. That is, a collection of $N_{VVT} \geq 1$ TrusteeCodebookCommitmentss, where N_{VVT} is the number of VoteVerificationTrustees for the Precinct indicated by (ElectionID, PrecinctID) .

$$\text{PrecinctCodebookCommitments} \doteq$$
$$(\text{TrusteeCodebookCommitments}_1, \ldots, \text{TrusteeCodebookCommitments}_{N_{VVT}})$$

Precinct ID : A UID for precincts. PrecinctIDs are required to be unique *within a fixed election*, but are not required to be universally unique. This allows PrecinctIDs to be reused over time by a jurisdiction.

Pre-Verification Code : A particular ElGamalPair returned to the Vote Client used to generate a VoteVerificationCode which is both voter specific and chosen answer specific.

Pre-Verification Codes : A vector of $\nu \geq 0$ PreVerificationCodes:

$$\text{PreVerificationCodes} \doteq (\text{PreVerificationCode}_1, \ldots, \text{PreVerificationCode}_\nu)$$

Pre-Verification Code Box : Each trustee generates a RawBallotBox with encrypted ElGamal pairs using his VoteVerificationKey and returns it inside a PreVerificationCodeBox structure.

Pre-Verification Code Boxes : A collection of PreVerificationCodeBoxes from all trustees.

Question Reference : A small integer, which, in the context of a fixed election is uniquely associated with a specific question. Question references *must* be assigned sequentially from 1 to NumBallotQuestions (0 to NumBallotQuestions - 1) since the PreVerificationCodes (or VoteVerificationCodes) will be returned in this order.

Question Text Structure : A specification according to a yet to be determined data standard of the human readable data (display text, title, shorthand name, etc.) associated with a ballot question (perhaps complicated to support multiple languages, etc.).

Random Bits : A collection of random, or pseudorandom bits.

Random Block : An array of RandomBits which may be generated by hashing certain seed values or may be generated by another method.

Random IJ State : An XML structure describing the current random numbers available. An attribute “SourceType” should be set to “PSEUDO” or “TRUE”, depending on whether one is generating pseudorandom or true random numbers. Indices i and j indicate the index of the first bit in the sequence.

$$\text{RandomIJState} \doteq (\text{RandomSeedKey})$$

or

$$\text{RandomIJState} \doteq ((i_0, j_0, n_0, bits_0)\,,\,\ldots\,,(i_m, j_m, n_m, bits_m))$$

Random Seed Key : A short byte sequence used to seed the random-number generator.

Random State : An XML structure describing the current random numbers available. An attribute "SourceType" should be set to "PSEUDO" or "TRUE", depending on whether one is generating pseudorandom or true random numbers. In the first definition, an attribute "Index" is included to indicate the location of the pointer in the RandomBlock.

$$\text{RandomState} \doteq \left\{ \begin{array}{c} \text{RandomSeedKey} \\ \text{RandomBlock} \end{array} \right\}$$

or

$$\text{RandomState} \doteq (\textsc{nil})$$

Raw Ballot Box : A vector of $B \geq 0$ *Raw Voted Ballots*

$$\text{RawBallotBox} \doteq (\text{RawVotedBallot}_1\,,\,\ldots\,,\text{RawVotedBallot}_B)$$

Raw Question Results : A pair

$$\text{RawQuestionResults} \doteq \left\{ \begin{array}{c} \textsc{question id} \\ (\text{MultiSetElement}_1\,,\,\ldots\,,\text{MultiSetElement}_Q) \end{array} \right\}$$

where the second element is a vector of multi-set elements – one for each allowed answer to the question.

Raw Results : A vector of question results

$$\text{RawResults} \doteq (\text{RawQuestionResults}_1\,,\,\ldots\,,\text{RawQuestionResults}_B)$$

Raw Voted Ballot : A vector of $m \geq 1$ *Voted Answers*:

$$\text{RawVotedBallot} \doteq (\text{VotedAnswer}_1\,,\,\ldots\,,\text{VotedAnswer}_m)$$

Result Verification Trustee : Currently a synonym for Trustee.

Secret Coefficients : Cryptographic quantities specific to the Key Sharing protocol.

$$\text{SecretCoefficients} \doteq (\theta_1\,,\,\ldots\,,\theta_t)$$

where

$$\theta \in \mathbf{Z}_q$$

Secondary Encryption Base : An additional element of the ElectionEncodingSubgroup used for ElectionVerification.

Secret Share : A modular integer (secret) with constraints based on the CryptoElectionParameters. The Secret Share, s for the authority with an TrusteeEvaluationPoint is and element of $\mathbf{Z}_q$ (where q is the ElectionSubgroupModulus) characterized by:

$$\text{SecretShare} \doteq s = f(\text{TrusteeEvaluationPoint}) =$$

$$\sum_{j=1}^{n} f_j(\text{TrusteeEvaluationPoint})$$

Seed Parameters : An XML string containing initial values for generating KeyGenParameters. The values indicate the number of Trustee objects to be created, the threshold number of Authorities to be used in Tabulation, and a seed for generating random numbers.

Shuffle Validity Proof : A zeroknowledge proof of correctness for shuffle.

Signature : A data string which may be used to ensure that another string was created, or endorsed, by a person whose GeneralPurposePublicKey we have.

Signed Ballot Box : The vector of Signed Voted Ballots that constitute input to tabulation. An authenticating signature (or vector of signatures) is appended for the purpose of "officially sealing" the ballot box. (The exact treatment of these signatures will be set as a matter of election policy.)

$$\text{SignedBallotBox} \doteq \left\{ \begin{array}{c} \text{ElectionID} \\ (\text{SignedVotedBallot}_1\,, \ldots\,, \text{SignedVotedBallot}_t) \\ (\text{Signature}_1\,, \ldots\,, \text{Signature}_I) \end{array} \right\}$$

Signed Blank Ballot : A *Blank Ballot* with an arbitrary number of detached signatures.

$$\text{SignedBlankBallot} \doteq \left\{ \begin{array}{c} \text{BlankBallot} \\ (\text{Signature}_1\,, \ldots\,, \text{Signature}_t) \end{array} \right\}$$

Signed Document : An XML structure containing a hash of the original plaintext to be signed, and a Signature.

Signed Election Parameters : The Crypto Election Prameters (CryptoElectionParameters), along with a (policy dependent) vector of authorizing signatures.

$$\text{SignedElectionParameters} \doteq \left\{ \begin{array}{c} \text{CryptoElectionParameters} \\ (\,\text{Signature}_1, \ldots, \text{Signature}_\rho\,) \end{array} \right\}$$

Signed Status Query Structure : A signed StatusQueryStruct.

Signed Status Response Structure : The format for "secure" replies from the Vote Kernel.

Signed Voted Ballot : A *Voted Ballot* with detached (voter) signature.

$$\text{SVB} \doteq \left\{ \begin{array}{c} \text{VotedBallot} \\ \text{Signature} \end{array} \right\}$$

Signed Voted Ballots : An XML structure representing a set of zero or more SignedVotedBallots. The order of the elements is irrelevant.

Signing Private Key : A key which can be used for signature generation.

Signing Public Key : A key which can be used for signature verification.

Status Query Structure : A (XML) structure for encoding a state, or status query passed to the Vote Kernal. This structure will likely include

- a "status type" enum to specify the type of query
- a challenge RandomBits (to prevent replays)
- an ElectionID (which may be NULL)
- a VoterID (which may be NULL)

Tabulation Threshold : A positive integer (t) which specifies the number of CommittedTrustees who must cooperate in order to tabulate the SignedBallotBox. It is determined as a part of Key Sharing and can not be changed thereafter. It must, by nature, satisfy $1 \leq t \leq n$, where n is the KeyShareWidth parameter in the CommittedTrusteeSet structure of the CryptoTabulationParameters structure contained in the CryptoElectionParameters structure of the BlankBallot.

Trustee : The data structure identifying an official or entity who has, in advance of the election, been appointed to "oversee" the election.

$$\text{Trustee} \doteq \left\{ \begin{array}{c} \text{Certificate} \\ \text{TrusteeEvaluationPoint} \end{array} \right\}$$

Trustee Codebook Commitments : A structure that represents all CodebookCommitments for a *fixed* ElectionID, *fixed* PrecinctID, and *fixed* VoteVerificationTrustee. That is, a collection of $N_{BSN} \geq 1$ CodebookCommitments, where N_{BSN} is the number of BallotSequenceNumbers for the Precinct indicated by (ElectionID, PrecinctID) .

$$\text{TrusteeCodebookCommitments} \doteq$$

$$\left\{ \begin{array}{c} \text{Trustee} \\ \text{BlankBallot} \\ (\text{BSNCodebookCommitments}_1, \ldots, \text{BSNCodebookCommitments}_{N_{BSN}}) \\ \text{Signature} \end{array} \right\}$$

Trustee Evaluation Point : A modular integer β where

$$\beta \in \mathbf{Z}_q^* = \mathbf{Z}_q - \{0\}$$

Trustee Partial Decrypt : The data structure representing the decryption information contributed by a *single* CommittedTrustee.

$$\text{TrusteePartialDecrypt} \doteq \left\{ \begin{array}{c} \text{CommittedTrustee} \\ \text{BallotBoxPartialDecrypt} \end{array} \right\}$$

Trustee Partial Decrypts : An XML structure representing a *set* of one or more PartiallyDecryptedBallotBox. The order of the elements of this set is irrelevant.

$$\text{TrusteePartialDecrypts} \doteq$$
$$\{\text{TrusteePartialDecrypt}_1 \,, \ldots \,, \text{TrusteePartialDecrypt}_t\}$$

Trustee Set : A set of Authorities.

$$\text{TrusteeSet} \doteq \{\text{Trustee}_1 \,, \ldots \,, \text{Trustee}_t\}$$

UID : Unique identification number.

UUID : Universally unique identification number.

Validated Voted Ballot : A *Signed* Voted Ballot (SignedVotedBallot) along with a vector of *Answer Validity Proofs* (AnswerValidityProof). The vector of AnswerValidityProofs should correspond directly with the RawVotedBallot in the SignedVotedBallot. That is, the RawVotedBallot is a vector of Voted Answers (VotedAnswer), and the i^{th} AnswerValidityProof should be a proper validity proof for RawBallotBox[i].

$$\text{ValidatedVotedBallot} \doteq \left\{ \begin{array}{c} \text{SignedVotedBallot} \\ \text{AnswerValidityProof}[m] \end{array} \right\}$$

(Note that the AnswerValidityProof array could be NULL on ballot submission if the voter wishes to opt out of VoterVerification.)

Vote Receipt : A VoteReceiptData object *signed* (by Vote Collection Agency)

$$\text{VoteReceipt} \doteq \left\{ \begin{array}{c} \text{VoteReceiptData} \\ \text{Signature} \end{array} \right\}$$

Vote Receipt Data : Data used for proof of voting. When signed (see VoteReceipt), can be used by voters to verify authenticity of their SignedVotedBallot in the election transcript (when it exists), and to mount a protest in case of discrepancy.

$$\text{VoteReceipt} \doteq \left\{ \begin{array}{c} \textsc{hash}(\text{SignedVotedBallot}) \\ (\text{PreVerificationCodes} \mid \text{VoteVerificationCodes}) \end{array} \right\}$$

Vote Signing Certificate : A Certificate corresponding to one of the VoteSigningKeys used in the election. In the case of remote voting, this Certificate is exactly a (registered) voter Certificate. In the case of poll site voting, all VoteSigningCertificates *must be published* for the purpose of election verification prior to the start of vote casting.

Vote Signing Certificates : A vector of VoteSigningCertificatess.

$$\text{VoteSigningCertificates} \doteq (\text{VoteSigningCertificate}_1, \ldots, \text{VoteSigningCertificate}_n)$$

Vote Signing Key : A SecretKey used for signing a voted ballot. In the case of remote voting, this is exactly the private key corresponding to a voter's Certificate. In the case of poll site voting, each voting machine must have a VoteSigningKey for the purpose of ballot encryption. Whether or not the same key is used for multiple machines is left as a policy decision.

Vote Verification Code : A data string. Depending on the underlying protocol, it may be computed from a PreVerificationCode.

Vote Verification Codes : A vector of $\nu \geq 0$ VoteVerificationCodes.

$$\text{VoteSigningCertificates} \doteq (\text{VoteVerificationCode}_1, \ldots, \text{VoteVerificationCode}_\nu)$$

Vote Verification Codebook : Data that assigns a VoteVerificationCode to each AnswerReference on the BlankBallotfor a fixed VoterID or BallotSequenceNumber. It is required that if $A_1 \neq A_2$ are two distinct AnswerReferences which are both possible responses to the *same* QuestionReference, Q, then the VoteVerificationCode for A_1 *must be different* then the VoteVerificationCode for A_2. However, A_1 and A_2

may sometimes share the same VoteVerificationCode if they are possible responses to different QuestionReferences. It should be noted that usually VoteVerificationCodebooks are computed from a collection of VoteVerificationCodebooks (or VoteVerificationKeys that represent them).

Vote Verification Codebook Share : Data generated by an individual VoteVerificationTrustee for the purpose of creating a VoteVerificationCodebook with shared trust characteristics. For the purpose of minimizing the amount of secret data that must be stored by each VoteVerificationTrustee, it is possible to associate a single VoteVerificationKey with a full set of VoteVerificationCodebookShares via a fixed pseudo-random process.

Vote Verification Key : A SecretKey used by an individual VoteVerificationTrustee to pseudo-randomly generate CodebookCommitments.

Vote Verification Keys : A vector of VoteVerificationKeys.

Vote Verification Statement : A statement provided to the voter after voting which contains VoteVerificationCodes corresponding to his selections.

Vote Verification Statements : A collection of VoteVerificationStatements.

Vote Verification Trustee : Currently a synonym for Trustee.

Voted Answer : An ElGamal pair encrypting the voter's chosen *Answer Mark*.

$$\text{VotedAnswer} \doteq \left\{ \text{ ElGamalPair } \right\}$$

Voted Ballot :

$$\text{VotedBallot} \doteq \left\{ \begin{array}{c} \text{ElectionID} \\ \text{VoterID} \mid \text{BallotSequenceNumber} \\ \textsc{hash}(\text{BlankBallot}) \\ \text{RawVotedBallot} \end{array} \right\}$$

Voter ID : UUID for voters.

Voter Roll : This data structure is needed for ballot authentication and deduplication. Essentially it is a vector of certificates (Certificate) corresponding to the set of eligible voters.

$$\text{VoterRoll} \doteq \left\{ \begin{array}{c} \text{Jurisdiction ID Info} \\ (\text{Certificate}_1\,, \ldots\,, \text{Certificate}_N) \end{array} \right\}$$

B Analysis of Detection Probabilities

An analysis of the detection confidence levels for various audit frequencies and electorate sizes will be made available in the near future.