ElectionGuard
Preliminary Specification v0.85

Josh Benaloh
Microsoft Research

Overview

This document describes the functionality of a toolkit that can be used in conjunction with many new and existing voting systems to enable both end-to-end (E2E) verifiability and privacy-enhanced risk-limiting audits (RLAs). The process involves a set of election trustees (often represented by canvassing board members) who generate keys in advance of each election and then use their keys after the conclusion of voting to decrypt election artifacts to enable public verification of the election tallies.

A device that collects votes (e.g., a ballot-marking device or optical scanner) calls the toolkit with the contents of each vote it receives. The toolkit uses the trustee public key(s) to encrypt each ballot, stores the encrypted ballot for later publication as part of a public record, and returns to the device a unique tracking code to be given to the voter (the tracking code is not used if only RLAs are performed).

At the conclusion of an election, each voting device uploads all of the encrypted ballots it has collected together with non-interactive zero-knowledge proofs that each item is an encryption of a legitimate ballot. These encrypted ballots are then homomorphically combined to form an aggregate encrypted ballot containing the election tallies. Election trustees then apply their keys to decrypt the tallies and provide a proof that the tallies are correct.

Observers can use this open specification and/or accompanying materials to write *election verifiers* that can confirm the well-formedness of each encrypted ballot, the correct aggregation of these ballots, and the accurate decryption of election tallies.

The details of the ElectionGuard Application Programming Interface (API) are included in a separate document. The principal purposes of this document are to specify the functionality of the ElectionGuard toolkit and to provide details necessary for independent parties to write election verifiers that consume the artifacts produced by the toolkit.

ElectionGuard Structure

This document describes the four principal components of ElectionGuard.

1. Baseline Parameters – These are general parameters that are standard in every election. An alternate means for generating parameters is described, but the burden of verifying

an election is increased if alternate parameters are used because a verifier would need to verify the proper construction of any alternate parameters.

2. Key Generation – Prior to each individual election, trustees must generate individual public-private key pairs and exchange shares of private keys to enable completion of an election even if some trustees become unavailable. Although it is preferred to generate new keys for each election, it is permissible to use the same keys for multiple elections so long as the set of trustees remains the same. A complete new set of keys must be generated if even a single trustee is replaced.
3. Ballot Encryption – While encrypting the contents of a ballot is a relatively simple operation, most of the work of ElectionGuard is the process of creating externally-verifiable artifacts to prove that each encrypted ballot is well-formed (i.e., its decryption is a legitimate ballot without overvotes or improper values).
4. Verifiable Decryption – At the conclusion of each election, trustees use their private keys to produce election tallies together with verifiable artifacts that prove that the tallies are correct.

Notation

In the remainder of this specification, the following notation will be used.

- $\mathbb{Z} = \{\dots, -3, -2, -1, 0, 1, 2, 3, \dots\}$ is the set of integers.
- $\mathbb{Z}_p = \{0, 1, 2, \dots, p-1\}$ is the additive group of the integers modulo p.
- $\mathbb{Z}_p^*$ is the multiplicative subgroup of $\mathbb{Z}_p$. When p is a prime, $\mathbb{Z}_p^* = \{1, 2, 3, \dots, p-1\}$.
- $\mathbb{Z}_p^r$ is the set of r^{th}-residues in $\mathbb{Z}_p^*$. Formally, $\mathbb{Z}_p^r = \{y \in \mathbb{Z}_p^*$ for which $\exists x \in \mathbb{Z}_p^*$ such that $y = x^r \bmod p\}$. When p is a prime for which $p - 1 = qr$ with q a prime that is not a divisor of integer r, then $\mathbb{Z}_p^r$ is an order q cyclic subgroup of $\mathbb{Z}_p^*$ and for each $y \in \mathbb{Z}_p^*$, $y \in \mathbb{Z}_p^r$ if and only if $y^q \bmod p = 1$.

Encryption

Encryption in ElectionGuard is performed using an exponential form of the ElGamal cryptosystem.[1] Primes p and q are publicly fixed such that q is not a divisor of $r = \frac{p-1}{r}$. A generator g of the order q subgroup $\mathbb{Z}_p^r$ is also fixed. (Any $g = x^r \bmod p$ for which $x \in \mathbb{Z}_p^*$ suffices so long as $g \neq 1$.)

A public-private key pair can be chosen by selecting a random $s \in \mathbb{Z}_q$ as a private key and publishing $K = g^s \bmod p$ as a public key.

---

[1] ElGamal T. (1985) A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In: Blakley G.R., Chaum D. (eds) Advances in Cryptology. CRYPTO 1984. Lecture Notes in Computer Science, vol 196. Springer, Berlin, Heidelberg. https://link.springer.com/content/pdf/10.1007/3-540-39568-7_2.pdf

A message $M \in \mathbb{Z}_p^r$ is then encrypted by selecting a random nonce $R \in \mathbb{Z}_q$ and forming the pair $(\alpha, \beta) = (g^R \bmod p, g^M \cdot K^R \bmod p)$. An encryption (α, β) can be decrypted by the holder of the secret s as

$$\frac{\beta}{\alpha^s} \bmod p = \frac{g^M \cdot K^R}{(g^R)^s} \bmod p = \frac{g^M \cdot (g^s)^R}{(g^R)^s} \bmod p = \frac{g^M \cdot g^{Rs}}{g^{Rs}} \bmod p = g^M \bmod p.$$

The value of M can be computed from $g^M \bmod p$ as long as the message M is limited to a small, known set of options.

Only two possible messages are encrypted in this way by ElectionGuard. An encryption of zero is used to indicate that an option is *not* selected, and an encryption of one is used to indicate that an option is selected.

Homomorphic Properties

A fundamental quality of the exponential form of ElGamal described above is its additively *homomorphic* property. If two messages V_1 and V_2 are respectively encrypted as $(\alpha_1, \beta_1) = (g^{R_1} \bmod p, g^{V_1} \cdot K^{R_1} \bmod p)$ and $(\alpha_2, \beta_2) = (g^{R_2} \bmod p, g^{V_2} \cdot K^{R_2} \bmod p)$, then the component-wise product $(\alpha, \beta) = (\alpha_1 \alpha_2 \bmod p, \beta_1 \beta_2 \bmod p) = (g^{R_1+R_2} \bmod p, g^{V_1+V_2} \cdot K^{R_1+R_2} \bmod p)$ is an encryption of the sum $V_1 + V_2$. (There is an implicit assumption here that $(V_1 + V_2) < q$ which is easily satisfied when V_1 and V_2 are both small. If $(R_1 + R_2) \geq q$, $(R_1 + R_2) \bmod q$ may be substituted without changing the equation since $g^q \bmod p = 1$.)

This additively homomorphic property is used in two important ways in ElectionGuard. First, all of the encryptions of a single option across ballots can be multiplied to form an encryption of the sum of the individual values. Since the individual values are one on ballots that select that option and zero otherwise, the sum is the tally of votes for that option and the product of the individual encryptions is an encryption of the tally.

The other use is to sum all of the selections made in a single contest on a single ballot. After demonstrating that each option is an encryption of either zero or one, the product of the encryptions indicates the number of options that are encryptions of one, and this can be used to show that no more ones than permitted are among the encrypted options – i.e., that more options were selected than permitted.

However, as will be described below, it is possible for a holder of a nonce R to prove to a third party that a pair (α, β) is an encryption of M without revealing the nonce R and without access to the secret s.

Non-Interactive Zero-Knowledge (NIZK) Proofs

ElectionGuard provides numerous proofs about encryption keys, encrypted ballots, and election tallies using the following four techniques.

1. A Schnorr proof[2] allows the holder of an ElGamal secret key s to interactively prove possession of s without revealing s.
2. A Chaum-Pedersen proof[3] allows an ElGamal encryption to be interactively proven to decrypt to a particular value without revealing the nonce used for encryption or the secret decryption key s. (This proof can be constructed with access to either the nonce used for encryption or the secret decryption key.)
3. The Cramer-Damgård-Schoenmakers technique[4] enables a disjunction to be interactively proven without revealing which disjunct is true.
4. The Fiat-Shamir heuristic[5] allows interactive proofs to be converted into non-interactive proofs.

Using a combination of the above techniques, it is possible for ElectionGuard to demonstrate that keys are properly chosen, that ballots are properly formed, and that decryptions match claimed values.

Baseline Parameters

Integer ElGamal encryption is used with a prime modulus (p) chosen such that $p - 1 = qr$ where q is a moderately-sized prime that is not a divisor of r. Because data confidentiality should be long-lived, the ElectionGuard default will use a 4096-bit prime p and a 256-bit prime q. A generator (g) of the order q multiplicative subgroup of $\mathbb{Z}_p^*$ is also provided along with $\bar{g} = g^{-1} \bmod p$. The principal reason for selecting integer ElGamal over elliptic curve ElGamal is the desire to make construction of election verifiers as simple as possible without requiring special tools or dependencies.

Standard parameters for ElectionGuard begin with the largest 256-bit prime $q = 2^{256} - 189$. The hexadecimal representation of q is as follows.

```
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFF43
```

---

[2] Schnorr C.P. (1990) Efficient Identification and Signatures for Smart Cards. In: Brassard G. (eds) Advances in Cryptology — CRYPTO' 89 Proceedings. CRYPTO 1989. Lecture Notes in Computer Science, vol 435. Springer, New York, NY. https://link.springer.com/content/pdf/10.1007%2F0-387-34805-0_22.pdf

[3] Chaum D., Pedersen T.P. (1993) Wallet Databases with Observers. In: Brickell E.F. (eds) Advances in Cryptology — CRYPTO' 92. CRYPTO 1992. Lecture Notes in Computer Science, vol 740. Springer, Berlin, Heidelberg. https://link.springer.com/content/pdf/10.1007%2F3-540-48071-4_7.pdf

[4] Cramer R., Damgård I., Schoenmakers B. (1994) Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols. In: Desmedt Y.G. (eds) Advances in Cryptology — CRYPTO '94. CRYPTO 1994. Lecture Notes in Computer Science, vol 839. Springer, Berlin, Heidelberg. https://link.springer.com/content/pdf/10.1007%2F3-540-48658-5_19.pdf

[5] Fiat A., Shamir A. (1987) How To Prove Yourself: Practical Solutions to Identification and Signature Problems. In: Odlyzko A.M. (eds) Advances in Cryptology — CRYPTO' 86. CRYPTO 1986. Lecture Notes in Computer Science, vol 263. Springer, Berlin, Heidelberg. https://link.springer.com/content/pdf/10.1007%2F3-540-47721-7_12.pdf

The modulus p is then set to be the largest 4096-bit prime which is one greater than a multiple of q. This works out to $p = 2^{4096} - 69q - 2650872664557734482243044168410288960$.

The hexadecimal representation of p is as follows.

```
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFBA
FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FE0175E3 0B1B0E79 1DB50299 4F24DFB1
```

The value of the cofactor r is then set to $r = \frac{p-1}{q}$, and $g = 2^r \bmod p$ is used as the generator of the order q multiplicative subgroup of $\mathbb{Z}_p^*$. The hexadecimal representation of g is as follows.

```
9B61C275 E06F3E38 372F9A9A DE0CDC4C 82F4CE53 37B3EF0E D28BEDBC 01342EB8
9977C811 6D741270 D45B0EBE 12D96C5A EE997FEF DEA18569 018AFE12 84E702BB
9B8C78E0 3E697F37 8D25BCBC B94FEFD1 2B7F9704 7F634232 68881C3B 96B389E1
34CB3162 CB73ED80 52F7946C 7E72907F D8B96862 D443B5C2 6F7B0E3F DC9F035C
BF0F5AAB 670B7901 1A8BCDEB CF421CC9 CBBE12C7 88E50328 041EB59D 81079497
B667B960 49DA04C7 9D60F527 B1C02F7E CBA66849 179CB5CF BE7C990C D888B69C
44171E4F 54C21A8C FE9D821F 195F7553 B73A7057 07263EAE A3B7AFA7 DED79ACF
5A64F3BF B939B815 C52085F4 0714F4C6 460B0B0C 3598E317 46A06C2A 3457676C
B345C8A3 90EBB942 8CEECEFA 6FCB1C27 A9E527A6 C55B8D6B 2B1868D6 EC719E18
```

9A799605 C540F864 1F135D5D C7FB62D5 8E0DE0B6 AE3AB90E 91FB9965 05D7D928

3DA833FF 0CB6CC8C A7BAFA0E 90BB1ADB 81545A80 1F0016DC 7088A4DF 2CFB7D6D

D876A2A5 807BDAA4 000DAFA2 DFB6FBB0 ED9D7755 89156DDB FC24FF22 03FFF9C5

CF7C85C6 8F66DE94 C98331F5 0FEF59CF 8E7CE9D9 5FA008F7 C1672D26 9C163751

012826C4 C8F5B5F4 C11EDB62 550F3CF9 3D86F3CC 6E22B0E7 69AC6591 57F40383

B5DF9DB9 F8414F6C B5FA7D17 BDDD3BC9 0DC7BDC3 9BAF3BE6 02A99E2A 37CE3A5C

098A8C1E FD3CD28A 6B79306C A2C20C55 174218A3 935F697E 813628D2 D861BE54

The inverse generator $\bar{g} = \frac{1}{g} \bmod p$ has the following hexadecimal representation.

7C3760F7 C5286704 4BCDE2D4 759615F1 69B873FC B465D96D BE3CBFA5 8AA5EA94

31FE08F7 AAC4F859 8C240BE6 194B03E3 7F8A9DC7 8A255A82 BCE95959 FF52A6DE

66CF240A 50EDB093 4A987FD9 DA4AFD73 A38011BD 08F4AE43 573BDD50 FA6F70EE

EA067D6E 57D446DE 9351BEE5 0E6AD9A5 B9282967 F1CDA890 A21C79C4 3C398755

9F415CCC 4E9E71C2 E0D7E4AA 95C23510 891F0C98 0D2F67DD 14EF589A 356D9FE7

79AD2288 5923FAAC 1D334EDC D64D1541 66446A96 879EEB61 D92ADB68 F7BFA1BA

F7F66B05 7409A10A 08297B79 31CDB706 21571E31 43335ED7 BF130C08 18A8F99D

60E71645 D399B793 11A28B7D 10D7F1D4 0918A836 1B937929 FB0E9B46 3F90E494

4E37EDBE 60F5F0BD 21F4737D D526B4FF 7EFE36EE 5C8A0456 3B8F04CF 8E7A29EE

9742DA6E 27B7442C 5E9BD207 3F6274ED FDD8CBEF 916F6433 19D8A385 D5D52587

25FC3FA8 ECCFE897 72C85A84 79754B8A 53A7F19E EB64A1BE 23A767D2 898F9152

91D680BC 8778462E 2A6490EC E23A5C99 F96F1677 3018050C 24D00A1C 720B05AC

6B74BDE8 1FDAF645 433A227E 75D13073 00DB62FA 259B711D 077923C1 23624482

C6CEEF6A 925FABA1 8E44A5C0 C02DF980 220B517B C210655A FD9A7C16 2A3FCFCE

FC12E7C9 1D625397 366B3570 596316E1 6DD24A1E 1DC330C0 F051A9C4 2E528E56

39750808 BEC8614C CA123F27 A76F043A 2FD7864E C61C4F66 3F896543 4A73E978

Alternative parameter sets are possible. A good source for parameter generation is appendix A of FIPS 186-4[6]. However, allowing alternate parameters would force election verifiers to recognize and check that parameters are correctly generated. Since these checks would be very different from other checks that are required of a verifier, allowing alternate parameters would add substantial complexity to election verifiers. For this reason, this version of ElectionGuard fixes the parameters as above.

An ElectionGuard version 1 election verifier may assume that the baseline parameters match the parameters provided above. However, it is recommended that the above parameters be checked against the parameters of each election to accommodate the possibility of different parameters in future versions of ElectionGuard.[7]

Key Generation

Before an election, the number of trustees (n) is fixed together with a threshold value (k) that describes the number of trustees necessary to decrypt tallies and election verification. The values n and k are integers subject to the constraint that $1 \leq k \leq n$. Canvassing board members can often serve the role of election trustees, and typical values for n and k could be 5 and 3 – indicating that 3 of 5 canvassing board members must cooperate to produce the artifacts that enable election verification. The reason for not setting k too low is that it will also be possible for k trustees to decrypt individual ballots.

Note that decryption of individual ballots does not directly compromise voter privacy since links between encrypted ballots and the voters who cast them are not retained by the system. However, voters receive tracking codes that can be associated with individual encrypted ballots, so any group that has the ability to decrypt individual ballots can also coerce voters by demanding to see their tracking codes.

Threshold ElGamal encryption is used for encryption of ballots. This form of encryption makes it very easy to combine individual trustee public keys into a single public key for encrypting ballots. It also offers a homomorphic property that allows individual encrypted votes to be combined to form encrypted tallies.

The trustees of an election will each generate a public-private key pair. The public keys will then be combined (as described in the following section) into a single election public key which is used to encrypt all selections made by voters in the election.

---

[6] NIST (2013) Digital Signature Standard (DSS). In: FIPS 186-4. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf

[7] If alternative parameters are allowed, election verifiers must confirm that p, q, r, g, and $\bar{g}$ are such that both p and q are prime (this may be done probabilistically using the Miller-Rabin algorithm), that $p - 1 = qr$ is satisfied, that q is not a divisor of r, and $1 < g < p$, that $g^q \bmod p = 1$, that $g\bar{g} \bmod p = 1$, and that generation of the parameters is consistent with the cited standard.

Ideally, at the conclusion of the election, each trustee will use its private key to form a verifiable partial decryption of each tally. These partial decryptions will then be combined to form full verifiable decryptions of the election tallies.

To accommodate the possibility that one or more of the trustees will not be available at the conclusion of the election to form their partial decryptions, the trustees will cryptographically share[8] their private keys amongst each other during key generation in a manner to be detailed in the next section. A pre-determined threshold value (k) out of the (n) trustees will be necessary to produce a full decryption.

Another parameter of an election should be a public ballot coding file. This file should list all of the contests in an election, the number of selections allowed for each contest, and the options for each contest together with associations between each option and its representation on a virtual ballot. It is assumed that each contest in the ballot coding file has a unique label and that within each contest, each option also has a unique label. For instance, if Alice, Bob, and Carol are running for governor, and David and Ellen are running for senator, the ballot coding file could enable the vector $\langle 0,1,0;\, 0,1\rangle$ to be recognized as a ballot with votes for Bob as governor and Ellen as senator. The detailed format of a ballot coding file will not be specified in this document. But the contents of this file are hashed together with the prime modulus (p), subgroup order (q), generator (g), number of trustees (n), decryption threshold value (k), date, and jurisdictional information to form a base hash code (Q) which will be incorporated into every subsequent hash computation in the election.

Overview of key generation

The n trustees of an election are denoted by $T_1, T_2, \ldots, T_n$. Each trustee T_i generates an independent ElGamal public-private key pair by generating a random integer secret $s_i \in \mathbb{Z}_q$ and forming the public key $K_i = g^{s_i} \bmod p$. Each of these public keys will be published in the election record together with a non-interactive zero-knowledge Schnorr proof of knowledge of possession of the associated private key.

The joint election public key will be

$$K = \prod_{i=1}^{n} K_i \bmod p.$$

To enable robustness and allow for the possibility of missing trustees at the conclusion of an election, the ElectionGuard key generation includes a sharing of private keys between trustees to enable decryption by any k trustees. This sharing is verifiable, so that receiving trustees can confirm that the shares they receive are meaningful; and the process allows for decryption without explicitly reconstructing private keys of missing trustees.

[8] Shamir A. How to Share a Secret. (1979) Communications of the ACM.

Each trustee T_i generates $k-1$ random polynomial coefficients $a_{i,j}$ such that $0 < j < k$ and $0 \leq a_{i,j} < q$ and forms the polynomial

$$P_i(x) = \sum_{j=0}^{k-1} a_{i,j} x^j \bmod q$$

by setting $a_{i,0}$ equal to its secret value s_i. Trustee T_i then publishes commitments $K_{i,j} = g^{a_{i,j}} \bmod p$ to each of its random polynomial coefficients. As with the primary secret keys, each trustee should provide a Schnorr proof of knowledge of the secret coefficient value $a_{ij,}$ associated with each published commitment $K_{i,j}$. Since polynomial coefficients will be generated and managed in precisely the same fashion as secret keys, they will be treated together in a single step below.

At the conclusion of the election, individual encrypted ballots will be homomorphically combined into a single encrypted aggregate ballot – consisting of an encryption of the tally for each option offered to voters. Each trustee will use its secret key to generate a partial decryption of each encrypted tally value, and these partial decryptions will be combined into full decryptions. If any election trustees are missing during tallying, any set of k trustees who are available can cooperate to reconstruct the missing partial decryption.

All spoiled ballots are individually decrypted in precisely the same fashion.

Details of key generation

Each trustee T_i in an election with a decryption threshold of k generates k polynomial coefficients $a_{i,j}$ such that $0 \leq j < k$ and $0 \leq a_{i,j} < q$ and forms the polynomial

$$P_i(x) = \sum_{j=0}^{k-1} a_{i,j} x^j \bmod q.$$

Trustee T_i then publishes commitments $K_{i,j} = g^{a_{i,j}} \bmod p$ for each of its random polynomial coefficients. The constant term $a_{i,0}$ of this polynomial will serve as the private key for trustee T_i, and for convenience we denote $s_i = a_{i,0}$, and its commitment $K_{i,0}$ will serve as the public key for trustee T_i and will also be denoted as $K_i = K_{i,0}$.

In order to prove possession of the coefficient associated with each public commitment, for each $K_{i,j}$ with $0 \leq j < k$, trustee T_i generates a Schnorr proof of knowledge for each of its coefficients as follows.

This Non-Interactive Zero-Knowledge (NIZK) proof proceeds as follows.

<u>NIZK Proof by Trustee T_i of its knowledge of secrets $a_{i,j}$ such that $K_{i,j} = g^{a_{i,j}} \bmod p$</u>

Trustee T_i generates random integer values $R_{i,j}$ in the range $0 \leq R_{i,j} < q$ and computes $h_{i,j} = g^{R_{i,j}} \bmod p$ for each $0 \leq j < k$. Using the hash function SHA-256 (as defined in NIST PUB FIPS

180-4[9]), trustee T_i then performs a single hash computation $c_i = H(Q, K_{i,0}, K_{i,1}, K_{i,2}, \ldots, K_{i,k-1}, h_{i,0}, h_{i,1}, h_{i,2}, \ldots h_{i,k-1}) \bmod q$ and publishes the values $K_{i,j}$, $h_{i,j}$, c_i, and $u_{i,j} = (R_{i,j} + c_i a_{i,j}) \bmod q$.

An election verifier should confirm both the hash computation of c_i and each of the $g^{u_{i,j}} \bmod p = h_{i,j} K_{i,j}^{c_i} \bmod p$ equations.

It is worth noting here that for any fixed constant α, the value $g^{P_i(\alpha)} \bmod p$ can be computed entirely from the published commitments as

$$g^{P_i(\alpha)} = g^{\sum_{j=0}^{k-1} a_{i,j}\alpha^j} \bmod p = \prod_{j=0}^{k-1} g^{a_{i,j}\alpha^j} \bmod p = \prod_{j=0}^{k-1} (g^{a_{i,j}})^{\alpha^j} \bmod p = \prod_{j=0}^{k-1} K_{i,j}^{\alpha^j} \bmod p.$$

Although this formula includes double exponentiation – raising a given value to the power α^j – in what follows, α and j will always be small values (bounded by n).

To share secret values amongst each other, it is assumed that each trustee T_i has previously shared a public encryption function E_i with the group.[10] Each trustee T_i then publishes the encryption $E_\ell\left(R_{i,\ell}, P_i(\ell)\right)$ for every other trustee T_ℓ – where $R_{i,\ell}$ is a random nonce.

Trustee T_ℓ can now decrypt each $P_i(\ell)$ encrypted to its public key and verify its validity against the commitments made by T_i to its coefficients $K_{i,0}, K_{i,1}, \ldots, K_{i,k-1}$ by confirming that the following equation holds:

$$g^{P_i(\ell)} \bmod p = \prod_{j=0}^{k-1} \left(K_{i,j}\right)^{\ell^j} \bmod p.$$

Trustees then publicly report having confirmed or failed to confirm this computation. If the recipient trustee T_ℓ reports not receiving a suitable value $P_i(\ell)$, it becomes incumbent on the sending trustee T_i to publish this $P_i(\ell)$ together with the nonce $R_{i,\ell}$ it used to encrypt $P_i(\ell)$ under the public key E_ℓ of recipient trustee T_ℓ. If trustee T_i fails to produce a suitable $P_i(\ell)$ and nonce $R_{i,\ell}$ that match *both* the published encryption and the above equation, it should be excluded from the election and the key generation process should be restarted with an alternate trustee. If, however, the published $P_i(\ell)$ and $R_{i,\ell}$ satisfy both the published encryption and the equation above, the receiving trustee T_ℓ should be excluded from the election and the key generation process should be restarted with an alternate trustee.

---

[9] NIST (2015) Secure Hash Standard (SHS). In: FIPS 180-4. https://csrc.nist.gov/publications/detail/fips/180/4/final

[10] A "traditional" ElGamal public key is fine for this purpose. But the baseline ElectionGuard parameters p and q are tuned for homomorphic purposes and are not well-suited for encrypting large values. The ElectionGuard trustee keys can be used by breaking a message into small pieces (e.g. individual bytes) and encrypting a large value as a sequence of small values. However, traditional public-key encryption methods are more efficient.

Once the baseline parameters have been produced and confirmed, all of the public commitments $K_{i,j}$ are hashed together with the base hash Q to form an extended base hash $\bar{Q}$ that will form the basis of subsequent hash computations. The hash function SHA-256 will be used here and for all hash computations for the remainder of this document.

Ballot Encryption

An ElectionGuard ballot is comprised entirely of encryptions of one (indicating selection made) and zero (indicating selection not made). To enable homomorphic addition (for tallying), these values are exponentiated during encryption. Specifically, to encrypt a ballot entry (V), a random value R is selected such that $0 \leq R < q$, and the following computation is performed.

- Zero (not selected) is encrypted as $(g^R \bmod p, K^R \bmod p)$.
- One (selected) is encrypted as $(g^R \bmod p, g \cdot K^R \bmod p)$.

Note that if multiple encrypted votes $(g^{R_i} \bmod p, g^{V_i} \cdot K^{R_i} \bmod p)$ are formed, their component-wise product $\left(g^{\sum_i R_i} \bmod p, g^{\sum_i V_i} \cdot K^{\sum_i R_i} \bmod p\right)$ serves as an encryption of $\sum_i V_i$ – which is the tally of those votes.[11]

A contest in an election consists of a set of options together with a selection limit that indicates the number of selections that are allowed to be made in that contest. In most elections, most contests have a selection limit of one. However, a larger selection limit (e.g., select up to three) is not uncommon in some elections. Approval voting can be achieved by setting the selection limit to the total number of options in a contest. Ranked choice voting is not supported in this version of ElectionGuard.[12] Also, write-ins are assumed to be explicitly registered or allowed to be lumped into a single "write-ins" category for the purpose of verifiable tallying. Verifiable tallying of free-form write-ins may be best done with a MixNet[13] design.

A legitimate vote in a contest consists of a set of selections with cardinality not exceeding the selection limit of that contest. To accommodate legitimate undervotes, the internal representation of a contest is augmented with "dummy" options equal in number to the selection limit. Dummy options are selected as necessary to force the total number of selections made in a contest to be equal to the selection limit. When the selection limit is one, for example, the single dummy option can be thought of as a "none of the above" option. With larger selection limits, the number of dummy options selected corresponds to the number of additional options that a voter could have selected in a contest.

---

[11] The initial decryption actually forms the value $g^{\sum_i V_i} \bmod p$. However, since $\sum_i V_i$ is a relatively small value, it can be effectively computed from $g^{\sum_i V_i} \bmod p$ by means of an exhaustive search or similar methods.

[12] Benaloh J., Moran. T, Naish L., Ramchen K., and Teague V. *Shuffle-Sum: Coercion-Resistant Verifiable Tallying for STV Voting* (2009) in Transactions of Information Forensics and Security.

[13] Chaum D. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms (1981) Communications of the ACM.

For efficiency, the dummy options could be eliminated in an approval vote. However, to simplify the construction of election verifiers, we presume that dummy options are always present – even for approval votes.

Two things must now be proven about the encryption of each vote.

1. The encryption associated with each option is either an encryption of zero or an encryption of one.
2. The sum of all encrypted values in a contest is equal to the selection limit for that contest (usually one).

The use of ElGamal encryption enables efficient zero-knowledge proofs of these requirements, and the Fiat-Shamir heuristic can be used to make these proofs non-interactive. Chaum-Pedersen proofs are used to demonstrate that an encryption is that of a specified value, and these are combined with the Cramer-Damgård-Schoenmakers technique to show that an encryption is that of one of a specified set of values – particularly that a value is an encryption of either zero or one. The set of encryptions of selections in a contest are homomorphically combined, and the result is shown to be an encryption of that contest's selection limit – again using a Chaum-Pedersen proof.

Note that the decryption of the selection limit could be more efficiently demonstrated by just releasing the sum of the nonces used for each of the individual encryptions. But, again to simplify the construction of election verifiers, a Chaum-Pedersen proof is used here as well.

The "random" nonces used for the ElGamal encryption of the ballot nonces are derived from a single 256-bit master nonce R_B for each ballot. For each contest listed in the ballot coding file, a contest nonce R_C is derived from the master nonce (R_B) and the contest label (L_C) as $R_C = H(L_C, R_B)$ using the hash function SHA-256. For each option listed in the ballot coding file, the nonce used to encrypt that option is derived from the contest nonce (R_C) and the selection label for that option (L_S) as $R = H(L_S, R_C)$.

A user of ElectionGuard may optionally provide an additional public key. If such a key is provided, ElectionGuard uses that key to encrypt each ballot's master nonce R_B and return this encryption together with the encrypted ballot.

Ballot nonces may be independent across different ballots, and only the nonces used to encrypt ballot selections need to be derived from the master nonce. The use of a single master nonce for each ballot allows the entire ballot encryption to be re-derived from the contents of a ballot and the master nonce. It also allows the encrypted ballot to be fully decrypted with the single master nonce.

Outline for proofs of ballot correctness

To prove that an ElGamal encryption pair (α, β) is an encryption of zero, the Chaum-Pedersen protocol proceeds as follows.

NIZK Proof that (α, β) is an encryption of zero (given knowledge of encryption nonce R)

The prover selects a random value u in the range $0 \leq u < q$ and commits to the pair $(a, b) = (g^u \bmod p, K^u \bmod p)$. A hash computation is then performed (using the Fiat-Shamir heuristic) to create a pseudo-random challenge value $c = H\big(\bar{Q}, (\alpha, \beta), (a, b)\big)$, and the prover responds with $v = (u + cR) \bmod q$. A verifier can now confirm the claim by checking that both $g^v \bmod p = a \cdot \alpha^c \bmod p$ and $K^v \bmod p = b \cdot \beta^c \bmod p$ are true.

NIZK Proof that (α, β) is an encryption of one (given knowledge of encryption nonce R)

To prove that (α, β) is an encryption of one, $\frac{\beta}{g} \bmod p$ is substituted for β in the above. The verifier can be relieved of the need to perform a modular division by computing $\beta \bar{g} \bmod p$ rather than $\frac{\beta}{g} \bmod p$. As an alternative, the verifier can confirm that $g^c \cdot K^v \bmod p = b \cdot \beta^c \bmod p$ instead of $K^v \bmod p = b \cdot \left(\frac{\beta}{g}\right)^c \bmod p$.

As with many zero-knowledge protocols, if the prover knows a challenge value prior to making its commitment, it can create a false proof. For example, if a challenge c is known to be forthcoming, a prover can generate a random v in the range $0 \leq v < q$ and commit to $(a, b) = \left(\frac{g^v}{\alpha^c} \bmod p, \frac{K^v}{\beta^c} \bmod p\right) = (g^v \alpha^{q-c} \bmod p, K^v \beta^{q-c} \bmod p)$. This selection will satisfy the required checks for (α, β) to appear as an encryption of zero regardless of the values of (α, β). Similarly, setting $(a, b) = \left(\frac{g^v}{\alpha^c} \bmod p, \frac{K^v g^c}{\beta^c} \bmod p\right) = (g^v \alpha^{q-c} \bmod p, K^v g^c \beta^{q-c} \bmod p)$ will satisfy the required checks for (α, β) to appear as an encryption of one regardless of the values of (α, β). This quirk is what enables the Cramer-Damgård-Schoenmakers technique to prove a disjunction of two predicates.

Sketch of NIZK Proof that (α, β) is an encryption of zero or one

After the prover makes commitments (a_0, b_0) and (a_1, b_1) to the respective assertions that (α, β) is an encryption of zero and (α, β) is an encryption of one, a single challenge value c is selected by hashing all commitments and baseline parameters. The prover must then provide challenge values c_0 and c_1 such that $c = c_0 + c_1 \bmod q$. Since the prover has complete freedom to choose one of c_0 and c_1, the prover can fix one value in advance – either c_0 if (α, β) is actually an encryption of one or c_1 if (α, β) is actually an encryption of zero. In response to the resulting challenge c, the prover uses this freedom to answer its faux claim with its chosen challenge value and then uses the remaining challenge value (as forced by the constraint that $c = c_0 + c_1 \bmod q$) to demonstrate the truth of the other claim. An observer can see that one of the two claims must be true but cannot tell which.

Details for proofs of ballot correctness

The full protocol proceeds as follows – fully divided into the two cases.

To encrypt an "unselected" option on a ballot, a random nonce r is selected uniformly from the range $0 \leq R < q$ and an encryption of zero is formed as $(\alpha, \beta) = (g^R \bmod p, K^R \bmod p)$.

NIZK Proof that (α, β) is an encryption of zero or one (given knowledge of encryption nonce R for which (α, β) is an encryption of zero)

To create the proof that (α, β) is an encryption of a zero or a one, randomly select c_1, v_1, and u_0 and form the commitments

$$(a_0, b_0) = (g^{u_0} \bmod p, K^{u_0} \bmod p)$$

and

$$(a_1, b_1) = \left(\frac{g^{v_1}}{\alpha^{c_1}} \bmod p, \frac{K^{v_1} g^{c_1}}{\beta^{c_1}} \bmod p\right) = (g^{v_1} \alpha^{q-c_1} \bmod p, K^{v_1} g^{c_1} \beta^{q-c_1} \bmod p).$$

A challenge value c is formed by hashing the extended base hash $\bar{Q}$ together with (α, β), (a_0, b_0), and (a_1, b_1) to form a challenge value $c = H\big(\bar{Q}, (\alpha, \beta), (a_0, b_0), (a_1, b_1)\big)$. The proof is completed by forming $c_0 = (c - c_1) \bmod q$ and $v_0 = (u_0 + c_0 \cdot R \bmod q)$ and answering the challenge by returning c_0, c_1, v_0, and v_1.

To encrypt a "selected" option on a ballot, a random nonce R is selected uniformly from the range $0 \leq r < q$ and an encryption of one is formed as $(\alpha, \beta) = (g^r \bmod p, g \cdot K^r \bmod p)$.

NIZK Proof that (α, β) is an encryption of zero or one (given knowledge of encryption nonce R for which (α, β) is an encryption of one)

To create the proof that (α, β) is an encryption of a zero or a one, randomly select c_0, v_0, and u_1 and form the commitments

$$(a_0, b_0) = \left(\frac{g^{v_0}}{\alpha^{c_0}} \bmod p, \frac{K^{v_0}}{\beta^{c_0}} \bmod p\right) = (g^{v_0} \alpha^{q-c_0} \bmod p, K^{v_0} \beta^{q-c_0} \bmod p)$$

and

$$(a_1, b_1) = (g^{u_1} \bmod p, K^{u_1} \bmod p).$$

A challenge value c is formed by hashing the extended base hash $\bar{Q}$ together with (α, β), (a_0, b_0), and (a_1, b_1) to form a challenge value $c = H\big(\bar{Q}, (\alpha, \beta), (a_0, b_0), (a_1, b_1)\big)$. The proof is completed by forming $c_1 = (c - c_0) \bmod q$ and $v_1 = (u_1 + c_1 \cdot R \bmod q)$ and answering the challenge by returning c_0, c_1, v_0, and v_1.

In either of the two above cases, what is published in the election record is the encryption (α, β) together with the commitments (a_0, b_0) and (a_1, b_1) which are all hashed together with the election's extended base hash to form the challenge value c which is published together with values c_0, c_1, v_0, and v_1.

An election verifier must confirm the following for each possible selection on a ballot:

- The given values α, β, a_0, b_0, a_1, and b_1 are all in the set $\mathbb{Z}_p^r$. (A value x is in $\mathbb{Z}_p^r$ if and only if x is an integer such that $0 \leq x < p$ and $x^q \bmod p = 1$ is satisfied.)
- The challenge c is correctly computed as $c = H\big(\bar{Q}, (\alpha, \beta), (a_0, b_0), (a_1, b_1)\big)$.
- The given values c_0, c_1, v_0, and v_1 are each in the set $\mathbb{Z}_q$. (A value x is in $\mathbb{Z}_q$ if and only if x is an integer such that $0 \leq x < q$.)
- The equation $c = c_0 + c_1 \bmod q$ is satisfied.
- The equations $g^{v_0} = a_0 \alpha^{c_0} \bmod p$, $g^{v_1} = a_1 \alpha^{c_1} \bmod p$, $K^{v_0} = b_0 \beta^{c_0} \bmod p$, and $g^{c_1} K^{v_1} = b_1 \beta^{c_1} \bmod p$ are all satisfied.

Proof of satisfying the selection limit

The final step in proving that a ballot is well-formed is demonstrating that the selection limits for each contest have not been exceeded. This is accomplished by homomorphically combining all of the (α_i, β_i) values for a contest by forming the aggregate contest encryption $(\alpha, \beta) = (\prod_i \alpha_i \bmod p\,, \prod_i \beta_i \bmod p)$ and proving that (α, β) is an encryption of the total number of votes allowed for that contest (usually one). The simplest way to complete this proof is to combine all of the random nonces R_i that were used to form each $(\alpha_i, \beta_i) = (g^{R_i} \bmod p, K^{R_i} \bmod p)$ or $(\alpha_i, \beta_i) = (g^{R_i} \bmod p, g \cdot K^{R_i} \bmod p)$ – depending on whether the value encrypted is zero or one. The aggregate nonce $R = \sum_i R_i \bmod q$ matches the aggregate contest encryption as $(\alpha, \beta) = (\prod_i \alpha_i \bmod p\,, \prod_i \beta_i \bmod p) = (g^R \bmod p, g^L K^R \bmod p)$ – where L is the selection limit for the contest. (Recall that L extra "dummy" positions will be added to each contest and set to one as necessary to ensure that exactly L selections are made for the contest.)

NIZK Proof that (α, β) is an encryption of L (given knowledge of aggregate encryption nonce R)

An additional Chaum-Pedersen proof of (α, β) being an encryption of L is performed by selecting a random U in the range $0 \leq U < q$, publishing $(a, b) = (g^U \bmod p, K^U \bmod p)$, hashing these values together with election's extended base hash $\bar{Q}$ to form a pseudo-random challenge $C = H\big(\bar{Q}, (\alpha, \beta), (a, b)\big)$, and responding by publishing $V = (U + CR) \bmod q$.[14]

Note that all of the above proofs can be performed directly by the entity performing the public key encryption of a ballot without access to the decryption key(s). All that is required is the nonces R_i used for the individual selection encryptions. This is sufficient to compute the effective aggregate nonce as $R = \sum_i R_i \bmod q$.

An election verifier must confirm the following for each contest on the ballot:

- The number of dummy positions matches the contest's selection limit L.
- The contest total (A, B) satisfies $A = \prod_i \alpha_i \bmod p$ and $B = \prod_i \beta_i \bmod p$ where the (α_i, β_i) represent all possible selections (including dummy selections) for the contest.

[14] One could simply release the aggregate nonce $R = \sum_i R_i \bmod (p - 1)$ to complete this proof. However, since Chaum-Pedersen proofs are being performed elsewhere, it is simpler for a verifier to just repeat the same steps.

- The given value V is in $\mathbb{Z}_q$.
- The given values a and b are each in $\mathbb{Z}_p^r$.
- The challenge value C is correctly computed as $C = H\big(\bar{Q}, (A, B), (a, b)\big)$.
- The equations $g^V = aA^C \bmod p$ and $g^L K^v = bB^C \bmod p$ are satisfied.

Tracking codes

Upon completion of the encryption of each ballot, a tracking code is prepared for each voter. The code is a running hash that begins with the extended base hash code $\bar{Q}$ and includes an identifier for the voting device, the location of the voting device, the date and time that the ballot was encrypted, and, of course, the encryption of the ballot itself. The hash (H) used for this purpose is SHA-256. The tracking code is formed as follows. $H_0 = H(\bar{Q})$ where $\bar{Q}$ is the extended base hash code of the election. For ballot with index $i > 0$, $H_i = H(H_{i-1}, D, T, B_i)$ where D consists of the voting device information described above, T is the date and time of ballot encryption, and B_i is an ordered list of the individual encryptions on the ballot – with the ordering as specified by the ballot coding file. At the conclusion of a voting period (this may be the end of a day in a multi-day election), the hash chain is closed by computing $\bar{H} = H(H_\ell, \text{"CLOSE"})$, where H_ℓ is the final tracking code produced by that device during that voting period. The close of the hash chain can be computed either by the voting device or subsequently by election administrators, and it is published as part of the election record.

An election verifier must confirm that each of the values in the running hash is correctly computed. Specifically, an election verifier must confirm each of the following.

- The equation $H_0 = H(\bar{Q})$ is satisfied.
- For each ballot B_i, $H_i = H(H_{i-1}, D, T, B_i)$ is satisfied.
- The closing hash $\bar{H} = H(H_\ell, \text{"CLOSE"})$ is correctly computed from the final tracking code H_ℓ.

Once in possession of a tracking code (*and never before*), a voter is afforded an option to either cast the associated ballot or spoil it and restart the ballot preparation process. The precise mechanism for voters to make these selections may vary depending upon the instantiation, but this choice would ordinarily be made immediately after a voter is presented with the tracking code, and the status of the ballot would be undetermined until the decision is made. It is possible, for instance, for a voter to make the decision directly on the voting device, or a voter may instead be afforded an option to deposit the ballot in a receptacle or to take it to a poll worker to be spoiled.

Verifiable Decryption

At the conclusion of voting, all of the ballot encryptions are published in the election record together with the proofs that the ballots are well-formed. Additionally, all of the encryptions of each option are homomorphically combined to form an encryption of the total number of times that option was selected. The encryptions (α_i, β_i) of each individual option are combined by forming the product $(A, B) = (\prod_i \alpha_i \bmod p\,, \prod_i \beta_i \bmod p)$. This aggregate encryption (A, B),

which represents an encryption of the tally of that option, is published in the election record for each option.

To decrypt an aggregate encryption (A, B), each available election trustee T_i computes its share of the decryption as

$$M_i = A^{s_i} \bmod p.$$

Each trustee T_i also publishes a Chaum-Pedersen proof of the correctness of M_i as follows.

NIZK Proof by Trustee T_i of knowledge of $s_i \in \mathbb{Z}_p^r$ for which both $M_i = A^{s_i} \bmod p$ and $K_i = g^{s_i} \bmod p$

Trustee T_i selects a random value u_i in the range $0 \leq u_i < q$ and commits to the pair $(a_i, b_i) = (g^{u_i} \bmod p, A^{u_i} \bmod p)$. The values (A, B), (a_i, b_i), and M_i are hashed together with the extended base hash value $\bar{Q}$ to form a challenge value $c_i = H(\bar{Q}, (A, B), (a_i, b_i), M_i)$, and trustee T_i responds with $v_i = (u_i + c_i s_i) \bmod q$.

An election verifier must confirm for each (non-dummy) option in each contest in the ballot coding file that the aggregate encryption (A, B) satisfies $A = \prod_j \alpha_j$ and $B = \prod_j \beta_j$ where the (α_j, β_j) are the corresponding encryptions on all cast ballots in the election record.

An election verifier must then confirm for each (non-dummy) option in each contest in the ballot coding file the following for each decrypting trustee T_i.

- The given value v_i is in the set $\mathbb{Z}_q$.
- The given values a_i and b_i are both in the set $\mathbb{Z}_q^r$.
- The challenge value c_i satisfies $c_i = H(\bar{Q}, (A, B), (a_i, b_i), M_i)$.
- The equations $g^{v_i} = a_i K_i^{c_i} \bmod p$ and $A^{v_i} = b_i {M_i}^{c_i} \bmod p$ are satisfied.

Decryption when all trustees are present

If all trustees are present and have posted suitable proofs, the next step is to publish the value

$$M = B \Big/ \left(\prod_{i=1}^{n} M_i \right) \bmod p.$$

This M has the property that $M = g^t \bmod p$ where t is the tally of the associated option.

In general, computation of this tally value t is computationally intractable. However, in this application, t is relatively small – bounded by the number of votes cast. Election administrators can determine this tally value t from M by exhaustive search, by precomputing a table of all possible M values in the allowable range and then performing a single look-up, or by a combination in which some exponentiations are precomputed and a small search is used to find the value of t (e.g., a partial table consisting of $g^{100} \bmod p$, $g^{200} \bmod p$, $g^{300} \bmod p$, … is precomputed and the value M is repeatedly divided by g until a value is found that is in the

partial table). The value t is published in the election record, and verifiers should check both that $M = g^t \bmod p$ and that $B = (M \cdot \prod_{i=1}^{n} M_i) \bmod p$.

Decryption with missing trustees

If one or more of the election trustees are not available for decryption, any k available trustees can use the information they have to reconstruct the partial decryptions for missing trustees as follows.

If trustee T_i is missing during decryption, each of at least k available trustees T_ℓ should use its share $P_i(\ell)$ of the secret value s_i previously shared by T_i to compute a share of the missing partial decryption M_i in the same way that it used its own secret s_ℓ. Specifically, trustee T_ℓ publishes partial decryption share

$$M_{i,\ell} = A^{P_i(\ell)} \bmod p.$$

Trustee T_ℓ also publishes a Chaum-Pedersen proof of the correctness of $M_{i,\ell}$ as follows.

NIZK Proof by Trustee T_ℓ of knowledge of $s_{i,\ell} \in \mathbb{Z}_p^r$ for which both $M_{i,\ell} = A^{s_{i,\ell}} \bmod p$ and $g^{s_i} \bmod p = \prod_{j=0}^{k-1} K_{i,j}^{\ell^j} \bmod p$

Trustee T_ℓ selects a random value $u_{i,\ell}$ in the range $0 \le u_{i,\ell} < q$ and commits to the pair $(a_{i,\ell}, b_{i,\ell}) = (g^{u_{i,\ell}} \bmod p, A^{u_{i,\ell}} \bmod p)$. The values (A, B), $(a_{i,\ell}, b_{i,\ell})$, and $M_{i,\ell}$ are hashed together with the extended base hash value $\bar{Q}$ to form a challenge value $c_{i,\ell} = H(\bar{Q}, (A, B), (a_{i,\ell}, b_{i,\ell}), M_{i,\ell})$, and trustee T_ℓ responds with $v_{i,\ell} = (u_{i,\ell} + c_{i,\ell} P_i(\ell)) \bmod q$.

It is important to note here that although the value $P_i(\ell)$ is known to both the missing trustee T_i and the trustee T_ℓ, it is not published or generally known. However, the value $g^{P_i(\ell)} \bmod p$ can be computed from public values as

$$g^{P_i(\ell)} \bmod p = \prod_{j=0}^{k-1} K_{i,j}^{\ell^j} \bmod p.$$

An election verifier must confirm for each (non-dummy) option in each contest in the ballot coding file the following for each missing trustee T_i and for each surrogate trustee T_ℓ.

- The given value $v_{i,\ell}$ is in the set $\mathbb{Z}_q$.
- The given values $a_{i,\ell}$ and $b_{i,\ell}$ are both in the set $\mathbb{Z}_q^r$.
- The challenge value $c_{i,\ell}$ satisfies $c_{i,\ell} = H(\bar{Q}, (A, B), (a_{i,\ell}, b_{i,\ell}), M_{i,\ell})$.
- The equations $g^{v_{i,\ell}} = a_{i,\ell} \cdot (g^{P_i(\ell)})^{c_{i,\ell}} \bmod p$ and $A^{v_{i,\ell}} = b_{i,\ell} {M_{i,\ell}}^{c_{i,\ell}} \bmod p$ are satisfied.

The final step to reconstruct a missing partial decryption M_i is to compute Lagrange coefficients for a set of k available trustees $\{T_\ell : \ell \in U\}$ with $|U| = k$ as

$$w_\ell = \frac{\left(\prod_{j\in(U-\{\ell\})} j\right)}{\left(\prod_{j\in(U-\{\ell\})}(j-\ell)\right)} \bmod q.$$

An election verifier should confirm that for each trustee T_ℓ serving to help compute a missing share of a tally, that its Lagrange coefficient w_ℓ is correctly computed by confirming the equation $\left(\prod_{j\in(U-\{\ell\})} j\right) \bmod q = w_\ell \cdot \left(\prod_{j\in(U-\{\ell\})}(j-\ell)\right) \bmod q$.

An election verifier should then confirm the correct missing tally share for each (non-dummy) option in each contest in the ballot coding file for each missing trustee T_i as $M_i = \prod_{\ell\in U}\left(M_{i,\ell}\right)^{w_\ell} \bmod p$.

Note that the missing secret s_i could be computed directly as $s_i = \sum_{\ell\in U} w_\ell P_i(\ell) \bmod q$. However, it is preferable to not release the missing secret and instead only release the partial decryption that the missing secret would have produced. This prevents the missing secret s_i from being used for additional decryptions without the cooperation of at least k trustees.

As an example, consider an election with five trustees and a threshold of three. If two trustees are missing at the time of decryption, the remaining three can perform any required decryptions by constructing missing partial descriptions as described in the text above. If, instead, they take the shortcut of simply reconstructing and then using the two missing secrets, then any of the three could, at a later time, use its own secret together with the two reconstructed secrets to perform additional decryptions without cooperation of any other trustees.

The final step is to verify the tallies themselves.

An election verifier should confirm the following equations for each (non-dummy) option in each contest in the ballot coding file.

- $B = M \cdot \left(\prod_{i=1}^{n} M_i\right) \bmod p$.
- $M = g^t \bmod p$.

An election verifier should also confirm that the text labels listed in the election record match the corresponding text labels in the ballot coding file.

Decryption of spoiled ballots

Every ballot spoiled in an election is individually verifiably decrypted in exactly the same way that the aggregate ballot of tallies is decrypted. Election verifiers should confirm all such decryptions so that casual observers can simply view the decryptions and confirm that they match their expectations.

An election verifier should confirm the correct decryption of each spoiled ballot using the same process that was used to confirm the election tallies.

An election verifier should also confirm that for each decrypted spoiled ballot, the selections listed in text match the corresponding text in the ballot coding file.

The Election Record

The record of an election should be a full accounting of all of the election artifacts. Specifically, it should contain the following.

- Date and location of an election
- The ballot coding file
- The baseline parameters
  - Primes p and q and integer r such that $p = qr + 1$ and r is *not* a multiple of q
  - A generator g of the order q multiplicative subgroup $\mathbb{Z}_p^*$
  - The multiplicative inverse $\bar{g}$ of g modulo p
  - The number n of election trustees
  - The threshold k of trustees required to complete verification
- The base hash value Q computed from the above
- The commitments from each election trustee to each of their polynomial coefficients
- The proofs from each trustee of possession of each of the associated coefficients
- The election public key
- The extended base hash value $\bar{Q}$ computed from the above
- Every encrypted ballot prepared in the election (whether cast or spoiled)
  - All of the encrypted options on each ballot (including "dummy" options)
  - The proofs that each such value is an encryption of either zero or one
  - The selection limit for each contest
  - The proof that the number of selections made matches the selection limit
  - The device information for the device that encrypted the ballot
  - The date and time of the ballot encryption
  - The tracker code produced for the ballot
- The decryption of each spoiled ballot
  - The selections made on the ballot
  - The cleartext representation of the selections
  - Partial decryptions by each trustee of each option
  - Proofs of each partial decryption
  - Shares of each missing partial decryption (if any)
  - Proofs of shares of each missing partial decryption
  - Lagrange coefficients used for replacement of any missing partial decryptions
- Tallies of each option in an election
  - The encrypted tally of each option
  - Full decryptions of each encrypted tally
  - Cleartext representations of each tally

    - Partial decryptions by each trustee of each tally
    - Proofs of partial decryption of each tally
    - Shares of each missing partial decryption (if any)
    - Proofs of shares of each missing partial decryption
    - Lagrange coefficients used for replacement of any missing partial decryptions
- Ordered lists of the ballots encrypted by each device

An election record should be digitally signed by election administrators together with the date of the signature. The entire election record and its digital signature should be published and made available for full download by any interested individuals. Tools should also be provided for easy look up of tracker codes by voters.

Applications to end-to-end verifiability and risk-limiting audits

The methods described in this specification can be used to enable either end-to-end (E2E) verifiability or enhanced risk-limiting audits (RLAs). In both cases, the ballots are individually encrypted and proofs are provided to allow observers to verify that the set of encrypted ballots is consistent with the announced tallies in an election.

In the case of E2E-verifiability, voters are given tracking codes to enable them to confirm that their individual ballots are correctly recorded amongst the set of encrypted ballots. In the case of RLAs, encrypted ballots are randomly selected and compared against physical ballots to obtain confidence that the physical records match the electronic records.

To support enhanced risk-limiting audits (RLAs), it may be desirable to encrypt the master nonce of each ballot with a simple administrative key rather than the heavyweight election encryption key. This streamlines the process for decrypting an encrypted ballot that has been selected for audit. It should be noted that the privacy risks of revealing decrypted ballots are substantially reduced in the RLA case since voters are not given tracking codes that could be used to associate them with individual ballots. The primary risk is a coercion threat (e.g., via pattern voting) that only manifests if the full set of ballots were to be decrypted.

While the administratively encrypted nonce can be stored in an electronic record alongside each encrypted ballot, one appealing RLA instantiation is for the administrative encryption of a ballot's nonce to be printed directly onto the physical ballot. This allows an RLA to proceed by randomly selecting an encrypted ballot, fetching the associated physical ballot, extracting the nonce from its encryption on the physical ballot, using the nonce to decrypt the electronic record, and then comparing the physical ballot contents with those of the electronic record. A malicious actor with an administrative decryption key would need to go to each individual physical ballot to obtain the nonces necessary to decrypt all of the encrypted ballots, and the access to do so would enable this malicious actor to obtain all of the open ballots without necessitating the administrative decryption key.

If E2E-verifiability and enhanced RLAs are both provided in the same election, there must be separate ballot encryptions (ideally, but not necessary, using separate election encryption keys) of each ballot. The E2E-verifiable data set must be distinguished from the enhanced RLA data set. Using the same data set for both applications would compromise voter privacy for voters whose ballots are selected for auditing.

Acknowledgements

The author is happy to thank Nicholas Boucher, Joey Dodds, Gerald Doussot, Aleks Essex, Chris Jeuell Luke Myers, Vanessa Teague, Aaron Tomb, Daniel Wagner, Jake Waksbaum, and Greg Zaverucha for many helpful comments and suggestions on earlier versions of this specification.