Distributed Homomorphic Voting

Trevor Henderson, Fernando Torija, Alex Noakes

May 12, 2016

Abstract

We devise and implement a scheme that allows a voter to vote at a polling station or supervised location and then allows the voter to leave the polling station and check that their vote has been counted correctly and that everyone in a given population group has voted once all while maintaining anonymity of choices for all voters.

We show a proof of concept in the polling scenario and discuss practicality for use now and in the future given legal, social, financial, and computing limitations and allowances. We propose possible solutions to some limitations and go on to talk about trade-offs between the various work-arounds in anonymity, security, transparency and error-catching.

1 Introduction

1.1 Voting

Voting is one of the cornerstones of modern governments across the world. In practice, nearly every country in the world holds elections in which at least some subset of the population cast a vote to make decisions about political leaders, laws and other bills that can range from fiscal appropriations, to declarations of war.[6] In the United States all adult citizens have the right to vote in federal elections including presidential, Senate, and House of Representatives elections. Apart from federal elections, citizens also vote in various state and local elections that can include town, county and special district elections for leaders in specific functions. Both state and federal elections are conducted by the state government in each state such that procedures vary from state to state.[1]

Voting is a unique task for a few reasons. First, voting is very important people because it has the potential to directly affect the decisions made by government. In addition, due to social pressures of elections, it is very important that voting systems allow a person to be completely anonymous in their choices. Lastly, it is important that each voter receive the exact share of voting influence which they are entitled. So, we see that voting as a task is very intolerant to tabulation errors, it must be able to prevent voter fraud so that people do not gain a greater share of influence in the outcome, it must be anonymous, and most importantly it must be transparent so that all voters can be sure that the system functions exactly as it says it does.

Electronic voting or e-voting encompasses a wide range of the use of electronic devices in the voting process. It includes voting solutions that automate anything from small tasks like marking choices on a paper ballot to full automation of registration, identification, vote cast, tallying, and production of fully tabulated results. Fully automated e-voting that performs according to the specifications laid out has many benefits from possibly reducing overall cost to a smaller environmental impact. The most important of these benefits is that electronic voting has the ability to offer complete transparency for public vote tallying and checking, complete vote anonymity, and a homogenous voting process for all people. We use this paper to describe an implementation of a system which provides all of those most important benefits.

For the purposes of this paper, we sought to implement a system that allows voters to cast a vote and verify within a certain population that 1) their own vote had been cast and added to the tally 2) every other person in the given population had voted only once all without compromising the anonymity between a voter and their choice (i.e. Alice being able to determine what Bob had voted). Before describing the implementation of the systems, we must lay the groundwork for the different cryptographic principles we utilize before delving into the intricacies of our system.

1.2 Mathematical Foundation

1.2.1 Homomorphic Encryption

With the introduction of fully homomorphic encryption by Gentry in 2009, the possible world of encryption applications was broadened. Our system uses homomorphic encryption, allowing the system to encrypt a single vote for a person into ciphertext where it is then sent to be posted and tabulated publicly.[2] Homomorphic encryption is a form of encryption that allows computations to be carried out on the ciphertext. Therefore, when the resulting ciphertext is decrypted, the result matches the result of the same computation on the original plaintext message. A traditional partially homomorphic encryption system is one that allows one or some types of operations to be carried out on the ciphertext. Whereas a fully homomorphic encryption scheme allows arbitrary computation on the ciphertext. The scheme does not fit into either of these categories. Our scheme allows for an arbitrary type of computation to be carried out. In this way it is unlike traditional methods of partial homomorphic encryption. On the other hand, our scheme has limited circuit depth. This means that only a limited number of operations can be performed until the ciphertext will not accurately decrypt into the identically computed plaintext. This limitation stems from the use of learning with errors to aid in encryption. This is a limitation we are willing to accept as it will not be the limiting factor in our system's scalability.[4, 8]

1.2.2 Learning with Errors

Learning with errors (LWE) is a problem that stems from the field of machine learning. It states that for (x, y) for $x \in \mathbb{Z}_q^d, y \in \mathbb{Z}_q, \epsilon$, in a linear function $f(x) + \epsilon \longmapsto y$ where ϵ is an arbitrary noise function, an algorithm will be able to recreate f. This problem is conjectured to be hard to solve. More recently, Regev (2005) has shown that LWE is at least as hard as several worst case lattice problems and has shown a reduction to the learning parity problem. We use these proofs as evidence that learning with errors is sufficiently difficult to use in our encryption problem. However, it is now clear as to why our scheme has limited circuit depth. By using an additive noise model on each ciphertext, will accumulate noise over operations and eventually, should enough noise accumulate in the message, we would no longer be able to recover the message text.[7]

2 Encryption Scheme Overview

We base our encryption scheme off of Gentry's 2013 Fully Homomorphic Scheme which relies on the LWE hardness problem[4] Our main contribution is modifying this scheme so that one can combine ciphertexts encrypted under different keys in order create ciphertexts encrypted under the sum of both keys, without knowing either key.

All of the operations we will be doing will be in the field $\mathbf{Z}_q$, where $q = 2^k$ for some integer k. We will defin n to be our lattice dimension, and $N = n \log_2 q$ to be our ciphertext dimension.

In our scheme a ciphertext $C \in \{0,1\}^{N \times N}$ encrypting message $\mu \in \mathbf{Z}_q$ has the form

$$CP\mathbf{v} = 2\mu P\mathbf{v} + \mathbf{e}$$

where $\mathbf{v} \in \mathbb{Z}_q^n$ is the secret key, $\mathbf{e} \in \mathbb{Z}_q^N$ is a small noise vector, and $P \in \mathbb{Z}_q^{N \times n}$ is a public invertible matrix that

a. Simultaneously diagonalizes C with every other ciphertext encrypted under the same public P, such that $P^{-1}CP = D$ where D is a diagonal matrix. In this way, all ciphertexts C commute.

b. Preserves the value of C under "flattening". We will describe a flattening operation $\textsc{Flatten}(C)$ which makes the entries of C small, while retaining that

$$\textsc{Flatten}(C)P = CP$$

2.1 Homomorphic operations

In order to add the values of two ciphertexts we simply add the ciphertexts and flatten the result

$$\textsc{Add}(C_1, C_2) = \textsc{Flatten}(C_1 + C_2)$$

Which returns the following

$$\begin{aligned} \textsc{Add}(C_1, C_2)P\mathbf{v} &= \textsc{Flatten}(C_1 + C_2)P\mathbf{v} \\ &= (C_1 + C_2)P\mathbf{v} \\ &= 2(\mu_1 + \mu_2)P\mathbf{v} + (\mathbf{e}_1 + \mathbf{e}_2) \end{aligned}$$

In order to multiply the values of a ciphertext by a constant like

$$\textsc{Mul}(\alpha, C) = \textsc{Flatten}(\textsc{Flatten}(\alpha \cdot I_N) \cdot C)$$

$$\begin{aligned}\textsc{Mul}(\alpha, C)P\mathbf{v} &= \textsc{Flatten}(\textsc{Flatten}(\alpha \cdot I_N) \cdot C)P\mathbf{v}\\ &= \textsc{Flatten}(\alpha \cdot I_N)CP\mathbf{v}\\ &= \textsc{Flatten}(\alpha \cdot I_N)(\mu P\mathbf{v} + \mathbf{e})\\ &= \alpha\mu P\mathbf{v} + \textsc{Flatten}(\alpha \cdot I_N)\mathbf{e}\end{aligned}$$

Note that if the error is initially bounded by B, addition increases that bound by a factor of 2 and multiplication increases it by a factor of N. The noise increase in both is independent of the message μ.

2.2 Ciphertext Combination

If ciphertexts C_1 and C_2, encrypting message μ under secret keys $\mathbf{v}_1$ and $\mathbf{v}_2$ respectively, they can be combined into a ciphertext C_{12} encrypting μ under $\mathbf{v}_1+\mathbf{v}_2$, without knowing either secret key. If $\mu = 0$, then $\textsc{Combine}_0(C_1, C_2) = \textsc{Flatten}(C_1C_2)$:

$$\begin{aligned}\textsc{Combine}_0(C_1, C_2)P(\mathbf{v}_1 + \mathbf{v}_2) &= C_1C_2P\mathbf{v}_1 + C_1C_2P\mathbf{v}_2\\ &= C_2(2\mu P\mathbf{v}_1 + \mathbf{e}_1) + C_1(2\mu P\mathbf{v}_2 + \mathbf{e}_2)\\ &= C_2\mathbf{e}_1 + C_1\mathbf{e}_2\end{aligned}$$

If $\mu = 1$ then $\textsc{Combine}_1(C_1, C_2) = \textsc{Flatten}(2C_1 + 2C_2 - C_1C_2 - 2I_N)$:

$$\begin{aligned}\textsc{Combine}_1(C_1, C_2)P(\mathbf{v}_1 + \mathbf{v}_2) =&2C_1P\mathbf{v}_1 + 2C_2P\mathbf{v}_2 + 2C_1P\mathbf{v}_2 + 2C_2P\mathbf{v}_1\\ &- C_1C_2P\mathbf{v}_1 - C_1C_2P\mathbf{v}_2 - 2P(\mathbf{v}_1 + \mathbf{v}_2)\\ =&2(2\mu P\mathbf{v}_1 + \mathbf{e}_1) + 2(2\mu P\mathbf{v}_2 + \mathbf{e}_2) + 2C_1P\mathbf{v}_2 + 2C_2P\mathbf{v}_1\\ &- C_2(2\mu P\mathbf{v}_1 + \mathbf{e}_1) - C_1(2\mu P\mathbf{v}_2 + \mathbf{e}_2) - 2P(\mathbf{v}_1 + \mathbf{v}_2)\\ =&2\mu P(\mathbf{v}_1 + \mathbf{v}_2) + 2(\mathbf{e}_1 + \mathbf{e}_2) - C_2\mathbf{e}_1 - C_1\mathbf{e}_2\end{aligned}$$

In the $\mu = 0$ case, the noise increases by a factor of $2N$ and in the $\mu = 1$ case the noise increases by a factor of $2N + 4$.

2.3 Flattening

We have made slight modifications to the operations described in [4] so that they work on matrices. The 4 operations we will use are BitDecomp, BitDecomp^{-1}, PowersOf2 and Flatten with the following properties for $X, Y \in \mathbf{Z}_q^{n\times n}$, $C \in \mathbf{Z}_q^{N\times N}$

$$\textsc{BitDecomp}(X) \cdot \textsc{PowersOf2}(Y) = XY$$

$$\begin{aligned}\textsc{Flatten}(C) \cdot \textsc{PowersOf2}(Y) &= X \cdot \textsc{PowersOf2}(Y) \\ &= \textsc{BitDecomp}^{-1}(X) \cdot Y\end{aligned}$$

$$\textsc{Flatten}(X) \in Z_{\{0,1\}}^{N\times N}$$

These functions and their properties form the basis of the encryption scheme. Intuitively, BitDecomp takes a matrix and decomposes each of elements bitwise to make a wider matrix. BitDecomp^{-1} undoes the BitDecomp operation, but is defined for general matrices not just bit matrices. Powersof2 takes each element a and vertically expands it into $\log_2 q$ elements $(a, 2a, \ldots, 2^{\log_2 q-1}a)$

Algorithm 1 PowersOf2$(X) = Y$

```
for each row $r$ and each column $c$ of $X$ do
    for $b = 0$ to $\log_2 q - 1$ do
        $Y[b + r\log_2 q][c] \leftarrow 2^b \cdot X[r][c]$
    end for
end for
```

Algorithm 2 BitDecomp$(X) = Y$

```
for each row $r$ and each column $c$ of $X$ do
    for $b = 0$ to $\log_2 q - 1$ do
        $Y[r][l + c\log_2 q] \leftarrow$ bit $b$ of $X[r][c]$
    end for
end for
```

Algorithm 3 $\textsc{BitDecomp}(X) = Y$

```
for each row $r$ and each column $c$ of $Y$ do
    $Y[r][c] \leftarrow \sum_{b=0}^{\log_2 q-1} 2^b \cdot X[r][l + c\log_2 q]$
end for
```

Algorithm 4 $\textsc{Flatten}(X) = Y$

```
return $\textsc{BitDecomp}(\textsc{BitDecomp}^{-1}(X))$
```

2.4 Construction of P

In order to construct P, we first generate a random invertible matrix $Q \in \mathbb{Z}_q^{n\times n}$. Then we can calculate $P = \textsc{PowersOf2}(Q)$ and $P^{-1} = \textsc{BitDecomp}(Q^{-1})$. Note that P^{-1} is the left inverse of P:

$$\begin{aligned} P^{-1} \cdot P &= \textsc{BitDecomp}(Q^{-1}) \cdot \textsc{PowersOf2}(Q) \\ &= Q^{-1}Q = I_N \end{aligned}$$

Since $P = \textsc{PowersOf2}(Q)$, then $\textsc{Flatten}(C)P = CP$ for all $C \in \mathbb{Z}_q^{N\times N}$

2.5 Encryption

To encrypt a message we first generate a base ciphertext X, an encryption of zero, and then use the homomorphic properties of X to encrypt other messages. To encrypt the base ciphertext, we first generate a random diagonal matrix $D \in \mathbb{Z}^{N\times N}$ with odd entries. Let $X = PDP^{-1}$. Note that all such X's are simultaneously diagonal under P and

$$\begin{aligned} \textsc{Flatten}(X)P &= \textsc{Flatten}(X) \cdot \textsc{PowersOf2}(Q) \\ &= X \cdot \textsc{PowersOf2}(Q) \\ &= XP \end{aligned}$$

So X remains simultaneously diagonal even if its flattened.

This X obeys the properties of ciphertexts but it has no associated secret key. We need to generate a secretkey $\mathbf{v}$ that satisfies:

$$XP\mathbf{v} = \mathbf{e}$$

Where $\mathbf{e}$ is some small error vector. To do this, we generate a random error vector $\mathbf{e}$, then compute

$$\mathbf{v} = D^{-1}P^{-1}\mathbf{e}$$

D must have an inverse modulo q since all of its entries are odd.

A message μ can then be encrypted by computing $C = Flatten(2I_N + X)$.

$$CP\mathbf{v} = (2\mu I_N + X)P\mathbf{v} = 2\mu P\mathbf{v} + \mathbf{e}$$

A ciphertext encrypting a message μ is indistinguishable from a ciphertext encrypting any other message without access to the secret key $\mathbf{v}$, since C is always of the form PDP^{-1}, where D has odd entries.

2.6 Decryption

In order to decrypt ciphertext C encrypting a message $\mu \in \{0, 1\}$ with secret key $\mathbf{v}$, the vector $P\mathbf{v}$ must have some component $x_i \in \left(\frac{q}{8}, \frac{3q}{8}\right)$ and the noise must be bounded $\|\mathbf{e}\|_\infty < q/4$. Let y_i be the ith component of $CP\mathbf{v}$. Then $\textsc{Dec}(C, \mathbf{v}) = \left\lfloor \frac{y_i}{2x_i} \right\rceil$. So if $y_i = 2\mu x_i + e_i$, then

$$\left\lfloor \frac{y_i}{2x_i} \right\rceil = \mu + \frac{e_i}{2x_i}$$

In order for the decryption to be valid it must be the case that $2\mu x_i < q$ and $\left|\frac{e_i}{2x_i}\right| < \frac{1}{2}$, which are both supported by the bounds above.

3 Voting Scheme

3.1 Initialization

To initialize a vote, voting officials need only generate and publicize P and P^{-1} as described above.

3.2 Vote Casting

Each user $i \in [1, m]$ encrypts their vote $\mu \in \{0, 1\}$ under two different secret keys $\mathbf{v}_i^1$ and $\mathbf{v}_i^2$. They publicize the corresponding ciphertexts C_i^1 and C_i^2 in

Algorithm 5 ChooseKeys

```
while true do
    Randomly generate a choice $\{c_1 \dots c_m\}$.
    $x \leftarrow \sum_{i=1}^{m} v_i^{c_i}$
    if $q/8 < x \leq 3q/8$ then
        return x
    end if
end while
```

addition to the first components of each secret key v_i^1 and v_i^2. We then wish to find a choice of ciphertext - secrey key pairs, $\{c_1 \dots c_m\}$, $c_i \in \{1, 2\}$ such that $\frac{q}{8} < \sum_{i=1}^{m} v_i^{c_i} \leq \frac{3q}{8}$. This can be done as follows:

We are assuming that v_i is distributed randomly enough that the sum of all v_i's can be treated as a random variable. In expectation this algorithm makes $O(4)$ iterations. The probability of failure is $(3/4)^k$, so this algorithm will terminate in a constant number of steps with high probability.

Let $C_i = C_i^{c_i}$ and $\mathbf{v}_i = \mathbf{v}_i^{c_i}$.

3.3 Vote Combination

We now have m ciphertexts $C_1 \dots C_m$ each of which encrypts vote μ_i under secret key $\mathbf{v}_1 \dots \mathbf{v}_m$ respectively. Each user must now combine their vote with each other vote to get a ciphertext C_i encrypted under $\mathbf{v}_1 \dots \mathbf{v}_m$. Since 0's and 1's are combined differently we must be careful when combining ciphertexts not to reveal too much about a users vote.

To deal with this a user will randomly turn half of the public ciphertexts into 0's and half of the ciphertexts into 1's. Then they will combine all 0's and 1's together as described above.

If C encrypts $\mu \in \{0, 1\}$, then $\textsc{Transform}_0(C) = \textsc{Flatten}(C(2I_N - C))$ encrypts 0.

$$\begin{aligned}
\textsc{Transform}_0(C)P\mathbf{v} &= C(2I_N - C)P\mathbf{v} \\
&= 2CP\mathbf{v} - CCP\mathbf{v} \\
&= 4\mu P\mathbf{v} + \mathbf{e} - C(2\mu P\mathbf{v} + \mathbf{e}) \\
&= 4\mu P\mathbf{v} + \mathbf{e} - 4\mu^2 P\mathbf{v} - 2\mu\mathbf{e} - C\mathbf{e} \\
&= \pm\mathbf{e} - C\mathbf{e}
\end{aligned}$$

Algorithm 6 SharedEncrypt(μ, i)

```
while true do
    $C_\mu \leftarrow C_i$
    $C_{1-\mu} \leftarrow 2\mu I_N$
    for ( do $j = 1$ to $m$, $j \neq i$)
        $b \leftarrow \{0,1\}^R$
        Combine$_b$($C_b$, Transform$_b$($C_i$))
    end for
    return Combine$_\mu$($C_\mu, 2I_N - C_{1-\mu}$)
end while
```

Similarly $\text{Transform}_1(C) = \text{Flatten}(2I_N + C(2I_N - C))$ encrypts 1.

In order for a user to decrypt such a vote, they would need to find exactly which messages were transformed into 0's and which were transformed into 1's, which takes 2^m guesses.

3.4 Vote Counting

Once encrypted votes are encrypted under $(\mathbf{v}_1 + \ldots + \mathbf{v}_m)$, then the votes can be counted by adding up all of the ciphertexts and flattening the result due to homomorphic additivity.

Since we can only decrypt bit messages, we will then multiply the result by $2(m)^{-1}$. The resulting bit is 1 if more than half of the m users voted for candidate 1 and 0 otherwise.

3.5 Decryption

After the votes are counted we will have a ciphertext C encrypting a bit represenging the winner encrypted under $\mathbf{v}_1 + \ldots + \mathbf{v}_m$. In the registration process we assured that the first entry of $P(\mathbf{v}_1 + \ldots + \mathbf{v}_m)$ is greater than $q/8$ and less than $3q/8$. If each user publishes the first entry of their decryption x_i of the product $CP\mathbf{v}_i$, we can sum all of the personal decryptions to compute the decryption:

$$\mu = \left\lfloor \frac{\sum_{i=1}^{m} x_i}{\sum_{i=1}^{m} v_i} \right\rceil$$

4 Security Analysis

This voting procedure assumes honest but curious voters. If a voter wanted to they could simply return random bits during the decryption step to make the decryption random. Additional measures would be taken to protect against this attack.

However regardless of what a user does they will not be able to decrypt a user's vote. Although an adversary can compute polynomially many encryptions under $(\mathbf{v}_1 + \ldots + \mathbf{v}_m)$, they will not be able to recover the secret key itself due directly to the Learning with Errors problem[7] Additionally, revealing the first entry of $\mathbf{v}$ does not reveal anything about $\mathbf{v}$ itself by the Leftover Hash lemma[9]. A users base ciphertext is information theoretic secure because for ciphertext encrypting C μ under $\mathbf{v}$, there is a secret key $\mathbf{v}'$ under which it encrypts $1 - \mu$. Ciphertext combination is secure so long as there are a large number of voters.

5 Runtime Analysis

Combining base ciphertexts requires $O(m)$ $N \times N$ matrix multiplications. The computation requires transforming ciphertexts into 0's and 1's which increases the noise by a factor of $(N+1)$, and combining all of the ciphertexts, which increases the noise by a factor of $(N+4)^{\lceil \log_2 m \rceil + 1}$.

The depth of the tallying computation is also $\lceil \log_2 m \rceil$, so there the noise increase by a factor of $N2^{\lceil \log_2 m \rceil}$. The additional factor of N comes from the last multiplication.

So the total error is $((N+1)N2^{\lceil \log_2 m \rceil}(N+4))^{\lceil \log_2 M \rceil + 1}B$ where B is the initial error bound. As long as

$$N(N+1)2^{\lceil \log_2 m \rceil}(N+4)^{\lceil \log_2 M \rceil + 1}B < \frac{q}{4}$$

then we can decrypt the message.

In order for the LWE problem to be hard B must be $O(2Sqrt(n))$ and in order to achieve k-bit security, m and n must be at least k[7]. To get a sense of perspective, let $n = m = 128$, achieving high security over a reasonable voting size. q would have to be approximately 2^{160}, meaning each integer would take up 20 bytes of memory. Ciphertexts would be arrays of size 20480×20480. With very heavy optimization multiplication of these ciphertexts would take on the order of minutes on a modern laptop.

6 Implementation

We have implemented all of the basic matrix operations described above in C++. The implementaion can be found on github. However it would take a significant amount of optimization - beyond the scope of this project - in order to get rid of constant factor overhead and approach the asymptotic runtime. So the implementation cannot handle our voting scheme even for a small number of people, however it does demonstrate the properties the system would have.

6.1 Inverse Modulo q

Finding the inverse of a matrix modulo q is not the same as finding one **R**. In our implimentation we took advantage of the following algorithm to find inverses [5]

Algorithm 7 Inverse(A)

```
    Find $X_0$, the modulo 2 inverse of $A$ by Gauss-Jordan Elimination
    $C \leftarrow I - X_0 A$
    return $(I + C^{2^i})(I + C^{2^{i-1}}) \dots (I + C)$ for $i \geq \log_2 \log_2 q$
```

7 Practicality

Due to long computation times on average computers, this scheme is not scalable to the level it would need to be for an election of a national scale. Voting would take a long time if the whole country did it on their own personal computer, and there is no guarantee that everyone will decrypt. Many people also have limited access to computers, and the computers they do have access to may be much less powerful than even the average computer.

Because of this, we discussed a new voting scheme that uses a combination of the current voting scheme and our electronic voting scheme. This new scheme would utilize the voting stations of the current scheme and instead place a voting terminal there. Each district would have a terminal that would have its own public key. Voters in this district would go to the terminal, input their vote, and the terminal will encrypt and decrypt the vote. Voters will be able to vote anonymously and ensure that they have been given their fair share of the vote. Because each station is now doing it's share of the decryption, much of work is taken off of the voter, and the system is still decentralized. It is important that this voting scheme remain decentralized, as the decentralization eliminates the chance of corrupt election officials, who may be trying to find out how people voted, or sway the vote in some way. Since the system is decentralized, all of the voting stations must collude together in order to break the security of our system, which becomes very unlikely as the number of stations increases.

8 Future Additions

Currently, this scheme assumes that voters are honest when they input their vote. For example, if a voter were to encrypt a 2 instead of a 1 or a 0 the decrypted result would turn into just noise. Since the result is not decrypted all at once, knowing whether the result is just noise is very difficult until the entire result has been decrypted. To fix this in the future, we could include zero knowledge proofs to ensure that voters are only encrypting either a 0 or a 1 and that they are being honest about their piece of the decryption.

9 Conclusion

Though this scheme, even with optimization, is currently not feasible for many reasons, eventually personal computers will be powerful enough such that an electronic voting scheme like this would work. Since we use a decentralized approach to homomorphic encryption, any worry of corrupt voting officials is eliminated; which makes this scheme better than a typical homomorphic electronic voting scheme, which would typically have some sort of trusted entity that would hold the key to decrypt the answer. Homomorphic electronic voting as a whole is currently the best way to achieve voter anonymity and eliminate voter coercion, voter fraud, and voter disenfranchisement. Our scheme comes with these benefits in addition to the benefits that stem from decentralization.

References

[1] "Comparative Data." *A.C.E Project.* Administration and Cost of Elections, 2015. Web. 11 May 2016. <http://aceproject.org/epic-en >.

[2] Gentry, Craig. *A Fully Homomorphic Encryption Scheme.* Diss. Stanford U. 2009.

[3] Gentry, Craig. "Fully Homomorphic Encryption Using Ideal Lattices." *STOC '09 Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing (2009):* 169-78. ACM Digital Library. ACM, 2009. Web. 11 May 2016.

[4] Gentry, Craig, Amit Sahai, and Brent Waters. "Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based." *Cryptology EPrint Archive.* 2013.

[5] Haramoto, H., and M. Matsumoto. "A P-adic Algorithm for Computing the Inverse of Integer Matrices." *Science Direct.* Hiroshima University, 2008. Web. 11 May 2016.

[6] Lynch, Tiffany. "The Country Thats Never Had an Election". *FP*. The Foreign Policy Magazine. 6 Nov. 2015. Web. 11 May 2016.

[7] Regev, Oded. "On Lattices, Learning with Errors, Random Linear Codes, and Cryptography." *STOC '05 Proceedings of the Thirty-seventh Annual ACM Symposium on Theory of Computing* (2005): 84-93. ACM Digital Library. ACM. 22 May 2005. Web. 11 May 2016.

[8] Rivest, Ron. "Public-Key Cryptography." 6.857: Computer and Network Security. MIT. 5 Apr. 2016. Lecture.

[9] Rubinfeld, Ronitt. "The Leftover Hash Lemma and Explicit Extractors." 6.895: Theory of Parallel Systems. MIT. Lecture. 30 April 2008. Web. 11 May 2015.