Secure Voting in the Cloud using Homomorphic Encryption and Mobile Agents

Mark A. Will*, Brandon Nicholson†, Marc Tiehuis† and Ryan K L Ko*
Cyber Security Lab
The University of Waikato
Hamilton, New Zealand
Email: *{willm, ryan}@waikato.ac.nz, †{bjn11, mat32}@students.waikato.ac.nz

***Abstract*—While governments are transitioning to the cloud to leverage efficiency, transparency and accessibility advantages, public opinion - the backbone of democracy - is being left behind. Statistics show that traditional paper voting is failing to reach the technological-savvy generation, with voter turnout decreasing every election for many first-world countries. Remote electronic voting is a possible solution facilitator to this problem, but it still faces several security, privacy and accountability concerns. This paper introduces a practical application of partially homomorphic encryption to help address these challenges. We describe a cloud-based mobile electronic voting scheme, evaluating its security against a list of requirements, and benchmarking performance on the cloud and mobile devices. In order to protect voter privacy, we propose moving away from a public bulletin board so that no individual cipher votes are saved, while still allowing vote verification. As the majority of the security threats faced by electronic voting are from the underlying system, we also introduce the novel concept of using a dedicated hardware server for homomorphic tallying and decryption.**

***Index Terms*—electronic voting; e-voting; mobile voting; homomorphic encryption; hardware model; cloud computing; privacy;**

I. Introduction

In New Zealand, the voter turnout at General Elections has been declining, as shown in Figure 1 [1]. This same trend has been seen in many other countries [2][3]. A survey in 2011, showed that 37% of non-voters either had other commitments or could not be bothered voting [4]. Governments have been looking at fixing this issue of modern democracy with remote e-voting (electronic voting) in the cloud [5]. Allowing votes to be cast at home over the internet with a personal device or mobile agent would make the process of voting simpler. The concept is easy to implement, but very difficult to secure. Estonia is currently using e-voting nation wide, even though it is not entirely secure [6][7].

Developing an e-voting scheme which is 100% secure against all attack vectors is incredibly difficult. But it does need to be as secure as current paper voting techniques. To protect voter privacy, votes must be encrypted so that no entity has knowledge of how someone voted. This is a perfect use case for homomorphic encryption [8], in particular partial homomorphic encryption. Because the result of a ballot is the sum of all votes for the available choices or candidates, additional homomorphic encryption is required. We focus on the Paillier [9] and Damgård—Jurik [10] Cryptosystems.

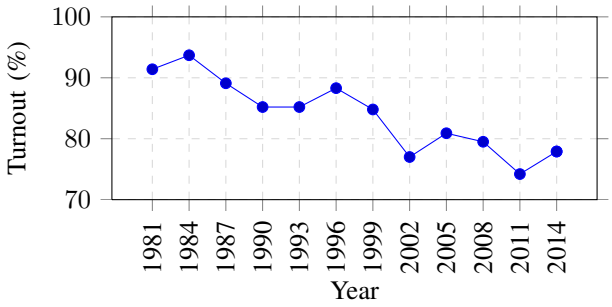

Fig. 1: New Zealand General Election Turnout.

In this paper we provide a list of requirements we believe should be used for evaluating any remote e-voting scheme. We propose a new technique for users to verify that their vote has been cast correctly, without the use of a public bulletin board. This is important because individual cipher votes should not be saved, otherwise votes can be revealed. We then analyse the performance of an e-voting scheme in the cloud using mobile agents. To improve the overall security of the system, we introduce the idea of using a hardware based tally and decryption server. We then evaluate our scheme before discussing current challenges and future work.

II. Related Work

A. Helios Voting Scheme

Helios is a web-based, open-audit voting system that was first proposed in 2008 [11]. The cryptography for the voting scheme itself is based on the work by Benaloh [12] and the Sako-Kilian mixnet [13]. To provide vote verification, Helios, as well as many other voting schemes [14][15][16][17][18][19], make use of a publicly visible bulletin board where each cipher vote is displayed, making it easier to trace back to the voter. The problem with using a bulletin board is that the organisation or personnel with access to the private key can view how individuals voted. Therefore this does not guarantee voter privacy. In order to verify the votes have been computed correctly, another authority needs to compute the result tally. This requires the private key, giving more entities access to individual vote details.

B. Self-Enforcing Voting Scheme

Hao *et. al.* defined "self-enforcing electronic voting" as a voting system that does not depend on any external tallying authorities [20]. Hence the system is self-tallying. This kind of voting scheme has a great advantage over more traditional schemes, as the trustworthiness of the system tends to increase when the human element is removed from the tallying process. The feasibility of such a system has been shown by Hao and Kreeger [14]. However, this scheme still relies on the use of public bulletin boards, which have their own vulnerabilities as mentioned above.

III. Country E-Voting Trials/Implementations

A. United Kingdom

The United Kingdom have been looking to utilise e-voting technology during general elections for many years. Their aim was to reduce cost, and increase voter turnout. At one stage they planned to have e-voting available after the 2005 general election. However the idea of introducing e-voting has been controversial, largely due to the negative publicity caused by electoral fraud incidents involving postal voting [2].

Despite this controversy, they continued to work towards their goal, conducting trials on a number of different e-voting systems during binding local elections in 2002/3 and 2007. However, only the 2007 trials included remote e-voting. In 2008, the United Kingdom concluded that the trials had not been overly successful. They have since suspended all e-voting trials indefinitely [21].

B. Switzerland

Switzerland has conducted a number of online voting trials in the cantons of Geneva and Zurich since 2004. The context of these trials has varied from student elections at the University of Zurich, to community-level referendums, all the way to cantonal and federal ballots. All ballots cast online in the trials were included along with the paper ballots as part of a binding result [3].

The systems used by Geneva and Zurich are similar. They mail out voting documentation prior to the election, which includes an e-voting card that lists a unique identification and PIN number. There have been no reports of security failures or manipulations of the e-voting systems during any of the trials. There have not yet been any large-scale trials conducted, so it is likely that flaws exist [3].

Therefore the trials so far have been considered a relative success. No noticeable technological problems have occurred, and the percentage of voters who cast their vote online has generally been high. However there is doubt that the use of e-voting systems have actually increased voter turnout, which like the United Kingdom, was one of their main goals [3]. At present, there appears to be no plans to trial online voting at a federal level.

C. Norway

In 2009, the Norwegian Government decided to start working towards *E-Valg 2011*, an e-voting pilot project for binding regional elections and local referendums in 2011. The key motivations for this system were to speed up the counting process, increase participation, and reduce the cost of running the elections and referendums [22].

The voter authentication mechanism that was utilised is MinID, a two-level authentication service that is used for various Norwegian government services. Using this system, the voter provides their credentials to the MinID server. Then the server generates a one-time password and sends it to the voter via SMS (Short Message Service). Once the voter enters this code, the authentication process is complete, and the voting process begins. After the vote has been cast, a verification code is generated by the voting server, and this is also sent to the voter by SMS. This gives the voter assurance that their vote has been cast as intended. The system may seem secure, however it relies heavily upon the assumption that the SMS channel is secure. This is not necessarily the case, as demonstrated by Koenig *et. al.* [23].

D. United States of America

Many states utilise some form of Internet voting, however the majority do not run trials that allow researchers to test the systems security. In 2010, Washington D.C. developed an Internet voting pilot system, intended to allow absentee voters to cast their vote via an online portal. Before using this system in the general election, they decided to launch a mock election, where they invited researchers to try and attack the system.

Wolchok *et. al.*, from the University of Michigan, approached the system from an attackers perspective (blackbox testing), by using only information available in the public domain. The test trials were planned to run over four days. However, it took less than 48 hours for Wolchok *et. al.* to take control of the election server, successfully changing every vote. They remained undetected, until deliberately leaving behind a clue for the conducting authority. Following this trial, the project was discontinued [24].

E. Canada

Canada is one of the leading countries in regards to Internet voting, offering more instances of online ballots in binding elections than any other country or jurisdiction [25]. For example, during the 2010 local elections in Ontario, more than 800,000 voters in 44 municipalities across the province were provided with the option of casting their ballot online. This is only one of many occasions on which Internet voting has been made available in Canada. Since 2003, when the concept was first introduced, about 60 municipalities across two provinces have successfully deployed Internet voting systems, intending to use them permanently [25].

At this stage there have been no Internet voting trials confirmed for provincial nor federal level elections, although Canadas national election agency has indicated that they wish to start looking at introducing Internet voting at the

federal election some time after 2015, pending parliamentary approval [25]. The idea of allowing the option of Internet voting during elections is also widely supported by the public, as indicated by a national survey conducted in 2011 which showed that about 85% of those surveyed were in favour of the idea [25].

Markham is the largest municipality in Ontario that offers Internet voting. It has done so during three consecutive binding elections in 2003, 2006 and 2010. As such it is one of the best examples to discuss. Prior to the election, eligible voters are sent a notification card by mail, which includes a unique, randomly generated PIN. They then register online, providing their PIN and date-of-birth as means of authentication. After choosing a passcode, the registration is complete. Finally, they will be sent out a second card by registered mail, which contains a new PIN. Once the Internet voting period (which usually goes for 5-7 days leading up to the election) begins, they can vote easily by providing their second PIN and the passcode that they chose during registration [25]. We were unable to find any studies relating to the security of these trials.

F. Estonia

Estonia first implemented Internet voting for their 2005 local office elections. They were the first country to introduce Internet voting on a nation scale. Since then, e-voting has been used for all local, regional and parliamentary elections [7].

When a person registers to vote in Estonia, they are issued with a smart identity (ID) card that contains a legal digital signature. This makes the voting process simple. Once the election begins, they can insert their smart ID card into a card reader attached to any computer. The voting application will then use this as a means of authentication [6][7].

A voter may cast their vote online at anytime during the voting period. Their vote can be changed as many times as they wish before the online voting period ends, which is usually the day before the election. On election day, if they decide to change their vote, they may also cast a paper vote. When the votes are tallied, an online vote will only count if they did not cast a paper vote [6].

There have been increasing concerns regarding the security of this system. But it was not until 2014 that Springall *et al.* from the University of Michigan conducted the first independent security analysis on the Estonia system. The evaluation comprised of both observations made throughout an election, and experimentation on the system itself. Their findings showed that although the system may have been secure and innovative when it was designed in the early 2000s, it was now fundamentally flawed. They were able to initiate client-side attacks that could silently steal votes from the voters computers. Server-side attacks were also discovered that could insert malware on the election servers, of which the implications could be disastrous. The team recommended that Estonia discontinue their e-voting system until there have been great advances in computer security, as they do not believe it can be made safe in today's environment [7].

IV. Cloud E-Voting System Requirements

We propose a coherent list of requirements that any proposed remote e-voting system should meet [26][27][28][29][30].

1) **Eligibility:** Only eligible voters are able to cast a vote.
2) **Unreusability:** Each voter may cast exactly one valid vote per ballot. Every ballot is distinct from the others and cannot be used more than once.
3) **Untraceability:** No ballot can be traced back to a specific voter by anyone else, including any authority.
4) **Verifiability:** Voters must be confident that their vote has been cast as intended.
5) **Tally Correctness:** All ballots are counted correctly. The total amount of all valid votes must not be more than the number of all registered voters. The tally process should be distributed across trusted authorities.
6) **Uncoerceability:** A voter cannot prove to anyone else how they cast their ballot. A briber/coercer cannot link any specific ballot to its owner.
7) **Auditability:** A voter may cast an invalid vote for the purposes of auditing.
8) **Accessibility:** Voters are not restricted by physical location from which they can cast their votes. Voters should not require special devices to enable them to vote. Voters should not be required to learn any new skills. Voters should be able to cast their ballots quickly without hassle.
9) **Fairness:** No one can know the intermediate results of the ballot.
10) **Soundness:** Malicious persons cannot figure out the intention of an eligible voter by intercepting any data transmitted between the voter and the voting authority. No voter can disturb or interrupt the voting process, such that they affect the result of the voting.
11) **Integrity:** The voting authority cannot control the election result by some malicious actions.
12) **Completeness:** All eligible voters are accepted to cast their vote and all ballots are counted correctly.

V. Homomorphic Schemes

The result of a ballot is usually the sum of all votes cast for something (e.g. candidate). Therefore the only operation required to calculate the result is addition. Instead of using fully homomorphic encryption, we can use a partially homomorphic scheme which supports addition. We say "no"$= 0$ and "yes"$= 1$ as these schemes require integer values [9][10]. For a scheme to support this simple e-voting scenario, it must meet the following requirements:

1) **Concealing:** Given any number of cipher values, it is infeasible to determine which value ("no" or "yes") they represent.
2) **Homomorphic-Tallying:** Given all the cipher values from a ballot, it is easy for anyone to computer the "yes" tally homomorphically.
3) **Verification:** A voter must be able to verify that their vote has been counted as intended towards the final tally.

4) **Multiple Candidates:** Ballots with multiple choices can be cast either by a single vote where each choice gets a bit range, or separate votes. Verification is required to guarantee x votes per voter.

A. Paillier Cryptosystem

Proposed in 1999 by Pascal Paillier [9], the Paillier cryptosystem is based on the problem that computing nth residue classes is computationally intensive. The encryption algorithm uses the public key $pk(n, g)$, and raises g to the unencrypted vote value m, multiplied by a random value r to the power of n, giving $E(m) = g^m r^n \mod n^2$. Therefore multiplying two cipher values, gives $c_1 c_2 = g^{(m_1+m_2)} r_1^n r_2^n \mod n^2$, giving a homomorphic addition operation. The limitation with this scheme is that we have to stay within modulus n^2. However given that the user should know the security parameters (i.e. number of voters), the keys can be generated accordingly.

B. Damgård—Jurik Cryptosystem

A generalisation of the Paillier Cryptosystem, the Damgård—Jurik Cryptosystem [10], allows the block length to be changed after the creation of the public keys. Damgård and Jurik also propose a threshold variant, which enables the secret key to be shared with x servers. However, each server cannot decrypt the cipher value alone. A large subset of the servers are required to work together to decrypt cipher values. The key property here is that if one server becomes compromised, by trying all combinations of servers for decryption, the compromised server can easily be flagged. Because the computation is distributed, each voter must send the cipher vote to each server. A zero-knowledge proof is also given where a prover can verify that a cipher value is in an allowed set of plaintext values. This is important because it can stop malicious voters from casting votes higher than 1. We use the Paillier Cryptosystem, but utilise the extra security provided with the distributed computation, and zero-knowledge proofs from the Damgård—Jurik Cryptosystem. This makes it more applicable to an e-voting system.

C. Partial Decryption

During key generation, the private key can be split into portions using concepts taken from Shamir's Secret Sharing algorithm. Each server is given an evaluation of a polynomial $P(x)$ which represents the private key. The i'th server gets a value $s_i = P(i)$, along with a verification value $v_i = v^{\Delta s i} \mod n^2$, where $\Delta = l!$, and v is a cyclic generator of squares in $\mathbb{Z}^*_{n^2}$. Each s_i is secret to each server.

In order to decrypt a given value, each decryption server must calculate its share as $c_i = c^{2\Delta s_i}$, where c is the ciphertext to decrypt. A zero-knowledge proof is constructed with each share to verify that it is indeed the case that the i'th server has computed their share correctly.

To combine the shares, we require w (chosen during key generation) or more to be correct in order to retrieve a successful decryption. We take a w subsets from $S = s_0, ...s_{l-1}$, and combine the elements of each in order to determine the correct combination.

$$c' = \prod_{i \in S} c_i^{2\lambda_i^S} \mod n^2 \qquad \text{where } \lambda_i^S = \Delta \prod_{\substack{i' < w \\ i' \neq i}} \frac{i'+1}{i'-i}$$

The result is of the form $c' = 4^{\Delta^2 d}$ so $c' = (n+1)^{4\Delta^2 m} \mod n^2$ with m the plaintext. It follows that applying $L(x) = \frac{x-1}{n} \mod n$ to c', and then multiplying by $(4\Delta^2)^{-1}$ results in retrieval of the desired plaintext m.

Note that in order to combine shares, it is only required to have knowledge of the public key. Therefore anyone can combine shares in order to decrypt the result, where only a small subset is required for a correct decryption.

D. Vote Proof

We wish to show that for a given $c = E(m, r) = (n+1)^m r^n \mod n^2$, m is encrypted by a value r. This implies that $c(n+1)^{-m} \mod n^2 = r^n \mod n^2$. In other words, if we can show that $c(n+1)^{-m} \mod n^2$ is an encryption of zero, a verifier should be convinced. If we have knowledge of the random value r which encrypts c, then we can always find a corresponding encryption of 0 for an n'th power as follows.

$$c = (n+1)^{kn} r^n \bmod n^2 = ((n+1)^k r)^n \bmod n^2, \quad k \in \mathbb{Z}$$

We then state a protocol to determine whether a given ciphertext u is an n'th power.

Proof of an n'th power:
Let P be a voter who wishes for their vote c to be verified by V. Let $u = E(0, v)$ be the vote containing P's vote. We then construct a proof for u as follows:

1) P chooses r at random in $\mathbb{Z}^*_n$ and calculates a as $E(0, r)$
2) By the Fiat-Shamir heuristic let h be a fixed hash function, P then calculates a value $e = h(a, ID(P))$ where $ID(P)$ is the unique id of P in this system.
3) Taking e as the challenge from V, P sends a, u and $z = rv^e \mod n$ to V.
4) V checks that u, a, z are co-prime to n and that $E(0, z) = au^e \mod n^2$ and if so, accepts u as being an encryption of some n'th power.

Using this protocol, we can then construct a proof which determines if an encryption contains one of two values, without disclosing which value it is. Given a vote c, and two possible values m_1 and m_2, compute $u_1 = c(n+1)^{-m_1} \mod n^2$ and $u_2 = c(n+1)^{-m_2} \mod n^2$. The prover is then required to show that one of u_1 or u_2 encrypt 0.

Proof of 1-out-of-2 n'th powers:
For this protocol, let $k_2 = k/2$ or $k_2 = 160$, where k is the bit size of the modulus n. Let P be a voter who wishes for their vote to be verified by V. Let u_1 and u_2 be constructed as mentioned prior, and determine the value v_1 for which $u_1 = E(0, v_1)$.

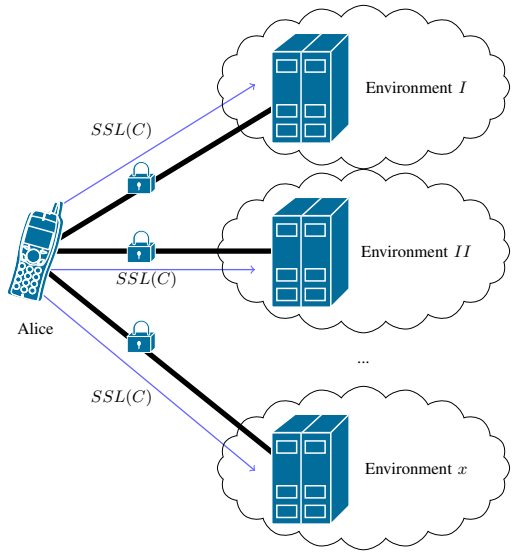

Fig. 2: Distributed Voting Model for the Cloud.

1) P constructs a proof of an n'th power on u_2, obtaining the values a_2, e_2, z_2. P then chooses r_1 at random in $\mathbb{Z}_n^*$ and computes $a_1 = E(0, r_1)$.
2) By the Fiat-Shamir heuristic let h be a fixed hash function, P then calculates the value $e = h(a_1, ID(P))$ where $ID(P)$ is the unique id of P in the system.
3) P computes $e_1 = e - e_2 \mod 2^{k_2}$ and $z_1 = r_1 v_1^{e_1} \mod n$ and sends $a_1, e_1, z_1, a_2, e_2, z_2$ to V.
4) V checks that $e = e_1 + e_1 \mod 2^{k_2}$, $E(0, z_1) = a_1 u_1^{e_1} \mod n^2$, $E(0, z_1) = a_2 u_2^{e_2} \mod n^2$ and that $u_1, a_1, z_1, u_2, a_2, z_2$ are co-prime to n and if so, accepts u_1 as being 1-of-2 values, m_1, or m_2.

We can use this protocol in our e-voting scheme to ensure that votes placed either count as a "yes", or as a "no", without exception.

VI. System Model

A. Key Generation

Currently key generation has to be performed by a single entity, which must be trusted to compute and distribute the shared keys to each server in the cloud. It is possible to compute the keys such that no single entity has knowledge of it [31], however this was out of scope for this paper.

B. Voting

Using a mobile agent, a user can vote by sending their cipher vote to each tally/decryption server used for the ballot as shown in Figure 2. Each connection between the user and a server should be encrypted over the Secure Sockets Layer (SSL) protocol. The tally and partial decryption servers should be spread across many organisations and regions. This reduces the probability of them all being compromised. However before a user can vote, they must be authorised/authenticated with each voting server. Depending on the ballots required security affects the level of authorised required. For example a simple ballot for a class president may only require a username and password. This could be made more secure with a confirmation string sent to the voter's email address, which they must include with their ID when voting. However for a government election, this is not secure enough.

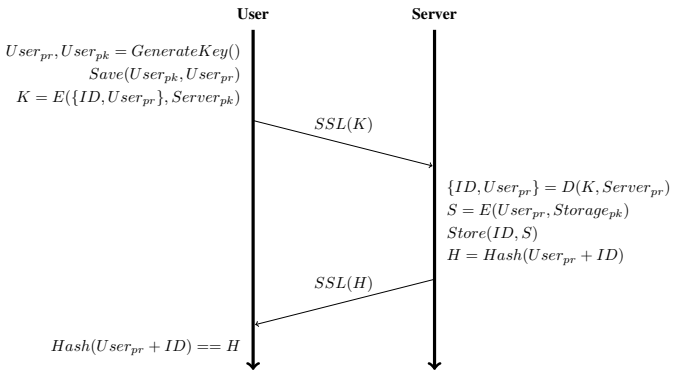

Fig. 3: User Submitting Verification Sequence

A solution that could be implemented is where every user has their own encryption key, which could be retrieved from any government agent (e.g. post or tax office). Therefore you must have proof of who you are before getting your unique key. During the ballot, all the servers have the decryptions keys for each voter. When voting, the cipher vote, vote proof and ID are encrypted with the voter's unique encryption key. This is then sent to the servers along with their ID in plain-text. To verify a voter, the server decrypts the cipher text using the user's decryption key retrieved by their ID. If the plain-text ID and cipher ID match, then the cipher vote and proof can be checked before the vote is added.

This however has an issue that at some point in time, the encryption and decryption keys for all users are visible. If a malicious user gains access to this data, the ballot could be compromised. Therefore only one key can be outside the user's device at all times. Instead of the government server creating all the verification keys, the users can generate them on their mobile device. They then encrypt the decryption key with each of the voting servers public keys (possibly at the government agent). The keys can then be transferred to the voting servers. Figure 3 shows a time sequence diagram for a user submitting their verification key. The same or different keys can be applied to every server. This could be enforced so that it must be done at a government agent, where the user can have their ID verified, the public keys of the servers can be checked, and a policy can be in place so that only government agents can access the voting servers during pre-voting. However this adds extra overhead to the voters. At any time prior to or during the ballot, a user can request a hash of their decryption key plus ID to verify it is stored correctly.

The IDs should be randomly generated at a set time before the ballot. Leading up to this point, voters are able to register if they have not already. Once the IDs are generated, the voter pool should be fixed. The list of IDs can then be sent to all the tally servers and the number of enrolled voters can be made public. After the ballot has finished, each tally server

can make the total number of voters and the number that actually voted available to the public. Publishing the number of enrolled voters twice helps prevent a fake user being added to the tally servers. This does not stop a fake user being added to the initial list of enrolled voters by a malicious employee. However this is an issue with current paper voting techniques as well.

There should not be a direct link between the users identity (e.g. name) and the ID they receive. However there does need to be a way to verify a user has the right to vote with the ID they received. Therefore the ID should be salted and hashed before being linked to the user. This allows a user to call/visit the authorities if they cannot vote, and verify that they are the owner of the ID. This also makes it much more challenging for a malicious employee to view all the IDs and who they are assigned too. This is important because if there is a high probability that an enrolled voter is not going to vote (e.g. they have passed away or left the country), it stops a corrupt insider from voting on their behalf (identity theft). Before the ID is hashed and therefore "lost", it must be printed then covered with scratch off ink, and sealed in an envelope with the voters postal address autonomously. At no point should the ID be visible to a human eye until the envelope is opened and the ink is removed. Sending out voting information via post has been successful in Switzerland's voting implementation [3]. For extra protection, there should be encryption between the printer and generation server, such that the ID and address are never stored in the printers memory in plain-text or visible to network sniffing.

In order for the server to trust the vote, a zero-knowledge proof is also required so that the server knows the vote is within the set $\{0, 1\}$. Without a proof, the voter can submit values > 1, making the ballot corrupt. If a ballot consists of multiple candidates/options, then the proof can be used to check that the sum of the votes equals 1 (proof requires access to all r values which the client has). For example if the ballot has three candidates, a voter can only choose one. Then the homomorphic addition of the three cipher votes must also be in the set $\{0, 1\}$. If it is outside this set, then the voter voted for more than one candidate. The voter would then send the 3 cipher votes with their proofs, along with a cipher vote consisting of the sum of the votes and a proof. This totals 4 cipher votes and 4 proofs. The server can check all proofs, and compute the homomorphic addition of the 3 votes to check that the 4^{th} cipher vote is correct.

The final voting sequence is given in Figure 4. Note that a user can only vote once, and this must be enforced in the tally server. This helps stop/identify if a malicious entity has voted for someone on their behalf. Either the user will vote and get an error because "they" have already voted, or the malicious entity will get an error if the user voted first. In the case where the user gets the error, they can contact the authorities to report the corrupt vote. A limitation of this design is that a corrupt vote cannot be removed since individual votes are not stored. Therefore there needs to be a policy in place such that if the number of corrupt votes reaches a certain threshold, or

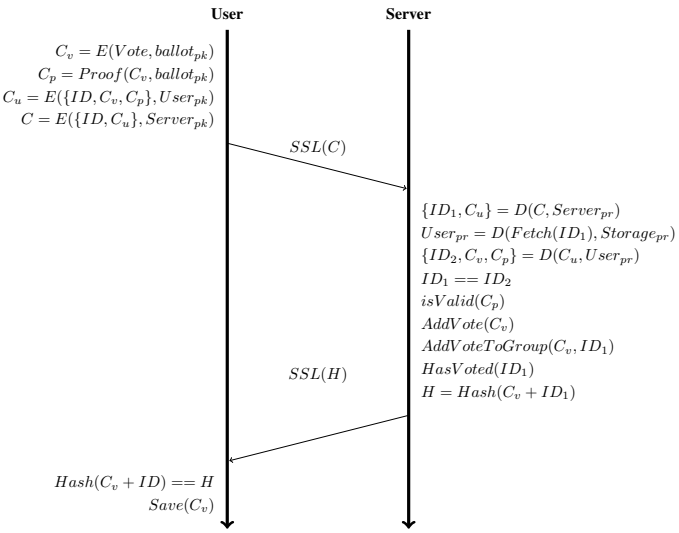

Fig. 4: User Voting Sequence

the difference between "yes" and "no" is under the number of corrupt votes, then the ballot is made invalid. However there is another solution to this problem. In Section VI-D we will cover splitting the voters into groups, where the group's tally is saved. Therefore only the voters in the group containing the corrupt vote can be asked to revote. How the voters are asked to revote is an open problem.

If a user does not wish to use the e-voting system, they can use paper voting. But their ID should be recorded as voted via paper. The electronic vote must be taken if both techniques were used, because the tally servers do not record each individual vote. Note that by allowing an electronic vote to be removed, the vote's value is revealed because of the difference in the final tally.

C. Tally Verification

The threshold variant of the Damgård—Jurik Cryptosystem which we adapted only requires a subset of all servers to provide a full decryption. Therefore to verify the final tally, anyone can request all the partial decryptions of the tally, and try all combinations. If none of the servers were compromised, then all the combinations will produce the same tally. However if a server was compromised, then not all the tallies will be the same value. With only one server compromised, it is easy to identify. But as the number of compromised servers increases, so does the difficulty of finding the correct tally. Therefore there must be a security parameter of the ballot which states at what point a revote is required.

D. Vote Verification

As discussed in Section II-A, using a public bulletin board does not guarantee individuals full vote privacy. Therefore we will propose a different technique which gives more protection to users' privacy, and gives voters the ability to check if their vote has been included in the tally.

Using the Paillier cryptosystem, in order for a voter to verify that their vote has been counted using only the result cipher

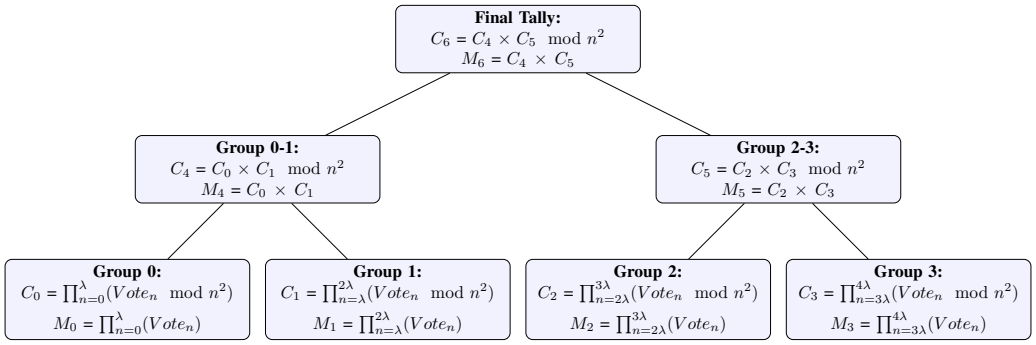

Fig. 5: Verifying a Paillier Based Voting Scheme

value is difficult. This is because it is within modulo n^2, which means checking that the user's cipher vote was part of the multiplication process is hard. Ideally the user would be able to save their cipher value. Then, after the voting is complete, multiply the result cipher value by the modular inverse of their cipher value. This would be equivalent to removing their vote from the tally. Note in this case after the voting anyone can decrypt the result as the decryption keys can be made public. In practice this is very challenging, because each cipher value would have to be a unique co-prime of n^2 in order to have a modular inverse.

A simpler solution is to combine the voters into groups of size λ. By combining votes, it is not possible to know how an individual voted. The user's cipher vote is homomorphically added to the group's tally. However we keep two values, one within modulo n^2, and another which is the result of the multiplication. For example, if $\lambda = 100$ and our key was 2048-bits, the modulo value would stay 2048-bits, but the other value would grow to 2048×100-bits worst case. We can then combine λ groups to form a tree. After the ballot is complete, each voter can verify by retrieving the non-modulo value from their group by dividing out their cipher value to check the difference in the tally. They can verify up to the final result by retrieving the next level of groups. Groups should be randomly assigned, because if they are assigned based on region, age or gender, there is a higher probability that all voters in the group will submit the same vote.

Figure 5 shows an example of the voting groups forming the verification tree. If a user from Group 1 wants to verify that their vote was counted once the voting has finished, they first request C_1 and M_1, along with the private key. We define C_1 as the groups sum value inside modulo n^2, and define M_1 as the groups sum value without the modulus operation. They then remove their vote $M_d = M_1/C_v$ and check that $Dec(M_1) - Dec(M_d) = Dec(C_v)$. To verify C_1, just check that $M_1 \mod n^2 = C_1$. Once the user knows that Group 1 contains their vote, they can check Group 0-1. They request C_4 and M_4 then check that $Dec(M_4) - Dec(M_4/C_1) = Dec(C_1)$ and $M_4 \mod n^2 = C_4$. The user could also request C_0 and M_0 to check the calculations for C_4, however it is assumed a user of Group 0 would verify this. These steps can be repeated until the final tally is verified to contain the user's vote.

With our proposed verification scheme, it is possible for a server to store individual votes as well as including them in a group. Then when the decryption key is made public, it is possible to map each vote to the voter. However this explanation has been using the Paillier Cryptosystem, when our System Model and Implementation use the threshold variant of the Damgård—Jurik Cryptosystem. We can use the same principle of vote verification, where each server keeps a M_x and C_x value for each group. Note that the groups of voters needs to be predefined so that each server uses the same mapping before voting begins.

To protect against malicious servers, the full decryption key cannot be made public. Because each server contains part of the decryption key, the user can request C_x, M_x and a part decryption of C_x. The user can then divide out their cipher vote giving $M_d = M_x/C_v$, then apply the modulus, $C_d = M_d \mod n^2$ (or n^{s+1}). To decrypt this value, the user sends C_d to each server and asks for a partial decryption. By combining all the partial decryptions of C_x, the vote tally for the group is given. Combining all the partial decryptions of C_d, should give the group tally minus the vote. To stop capturing of the partial decrypted values (e.g. network sniffing or man-in-the-middle attacks), the user sends two C_d values, one real and the other a dummy. The first is where the vote is "no", and the second where the vote is "yes". Therefore the user can verify their vote, but any entity capturing the partial decryptions does not know which is the user's vote. Note that when a user sends back the C_d values, they must be authenticated, and have a limitation on verifications to help prevent denial-of-service attacks. Also, given a small group size, the probability of a voter getting a false positive for verification is very low.

Allowing a voter to decrypt values after the ballot at first glance appears to be a security flaw, because anyone can decrypt cipher voters. However the challenge is capturing the cipher votes. In Section VIII we address the issue of a malicious user on the server capturing cipher votes, and prevent eaves-dropping or man-in-the-middle attacks. That just leaves the mobile agent since it needs to store the cipher vote for verification. We discuss the security issues of mobile agents in Section VII-B2. To summarise, if the mobile agent can be made secure, then the verification is also secure.

Example:
In this example, a class of 8 students are asked to vote on whether they like their teacher. The servers split the class into 2 groups, where Group 0 contains students 0-3 and Group 1 contains students 4-7. The students made the following votes using a 32-bit public key, giving a final tally of 4 votes for "yes".

Voter 0	85108103601502032158
Voter 1	80478975519969916818
Voter 2	82205080082953515845
Voter 3	68976771831363902506
Voter 4	69855354796941217103
Voter 5	41321137482081815425
Voter 6	14280272528528046683
Voter 7	86056116539738524768

Alice is student 1, and would like to verify her vote of "yes" has been included in the final tally. She requests the M and C values for her group from any or all of the servers. She then requests all partial decryptions of her groups C value. In this example her group's tally was 2. For this example we will not check C_0.

$$M_0 = 472450396734543875902171716038368870676253244103133400909464$$
$$C_0 = 35718563236129925271$$

$$M_d = M_0 / C_{vote}$$
$$M_d = 587048224312088607322668475616438878794 8$$

$$C_d = M_d \mod n^2$$
$$C_d = 10645547762907569226$$

She then sends C_d to all the servers for a partial decryption (and dummy C_d), which results in a value of 1. Therefore because $2 - 1 = 1$, Alice's vote was counted for her group's tally. Alice can then verify that her group's tally is included in the final tally using the same process.

Bob is student 2, but cannot remember if he submitted his "yes" vote on time or not. He follows the same steps Alice did. The decryption of the difference gives 7175337003, which means $2 - 7175337003 \neq 1$. Therefore Bob's vote did not count towards the final tally.

VII. Implementation

A. Partially Homomorphic C Library

Currently there does not exist a library to help build applications in the cloud which would like to use partially homomorphic encryption. Therefore in this paper we also propose the use of *libhcs* [32], a C library which implements partially homomorphic schemes. The goal of *libhcs* is to bring homomorphic encryption to the cloud, by making them easy to use for cloud developers. Currently we have implemented the Paillier [9], Damgård—Jurik [10] and El-Gamal [33] cryptosystems. The library can also be compiled to support mobile chipsets, provided that GMP (The GNU Multiple Precision Arithmetic Library) [34] is also supported.

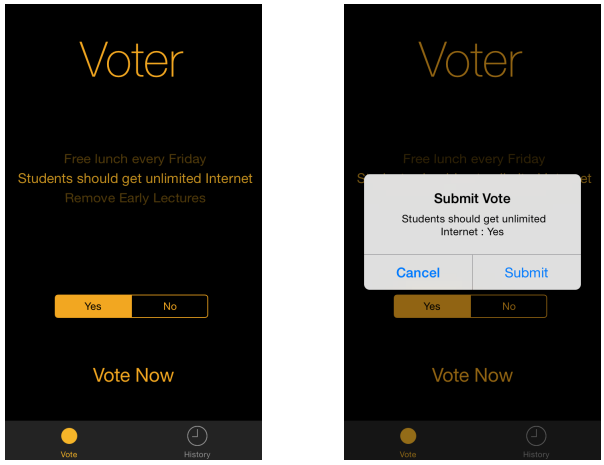

Fig. 6: Simple Voting Mobile Application Example.

B. Mobile Implementation

1) Performance: For our performance testing, we developed a simple iOS application to be run on an iPhone 5. The iPhone 5 was chosen because it is the one of the last iPhones to have a 32-bit processor, and is still one of most used smartphones [35]. The example application is shown in Figure 6, where a vote is being cast. This application makes use of libhcs [32] and GMP [34] for computing the cipher vote. How the ballots (questions in this case) are added to the application can be done in many ways. It could be that the user must manually enter the voting servers which can send updates on ballots. This would be sufficient for smaller ballots like at a university. However for a government election, the servers and ballots may need to be hardcoded into the application. Therefore because the application is signed in the store [36][37], a user can verify they received the right information.

We first tested the performance on an iPhone 5 (1.3 GHz dual core Apple A6) for encrypting a vote with different bit size for N. Note that if N is 32-bits, then because the cipher value is modulo N^2, the result can get up to 64-bits in size. Figure 7 below shows that we get an exponential curve, which is expected. This is because as the bit size increases, the number of instructions to handle a single operation increases. A proof is also required for the cipher value so that the tally server knows it is within a set of allowed values. The performance for generating the proof as shown below in Figure 7, is even worse than encryption. Section III details that many of the goals for adopting e-voting was to get more people to vote. Having a quick voting time would help to accomplish this goal. Given the results of encryption and proof generation in Figure 7, with current mobile technologies the largest size of N is 2048-bits. When taking into account network transmission times, making a voter wait for less than 30-seconds is much better than the time to travel to a polling station. For smaller scale ballots, using $size(N) = 1024$-bits would be even faster.

For verification, a user's device must divide out their cipher vote from their large group value as discussed in Section VI-D.

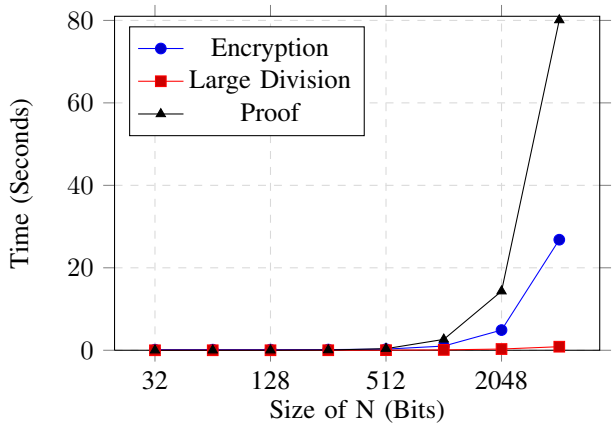

Fig. 7: Mobile Voting Performance on an iPhone 5.

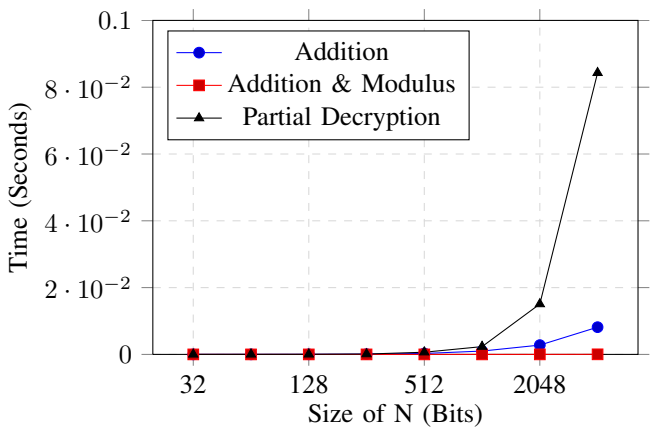

Fig. 8: Voting Performance for a Server Implementation.

We tested the time to complete a large division instruction on the iPhone 5 for groups of 100 voters. Figure 7 once again shows the time increases with the bit size. With $size(N) =$ 4096-bits, the instruction (8192×100-bits / 8191-bits) takes less than 0.9 seconds. Therefore the cost of this instruction is manageable given the extra voter privacy it provides.

2) Security Threats: A current issue of using mobile agents for voting is that mobiles are not secure [38][39][40][41]. The two biggest mobile operating systems [35], Android and iOS, must be secure for mobile voting to become successful. However both have security flaws and vulnerabilities at this stage. Another issue is that the user must trust the application, which could be achieved by open sourcing the code, and having the application signed by a trusted authority.

With the ever growing amount of malware for Android devices, private information such as messages and accounts can be leaked from infected devices [40]. Malware could therefore be used to leak the user's ID and their private encryption key generated for proving their identity. Unlike Google, Apple does a relatively better job of stopping malware being uploaded to their AppStore [39]. However this does not mean Apple devices are secure enough for government voting, especially ones that have been jailbroken [39]. For example device backups to online storage are another threat for data leakage. However when compared to Android, iOS has a better security model, especially the permission system. Android blacklists permissions, such that you can only use the application if you accept all the permissions, whereas iOS whitelists permissions, and does allow an application to be used even with some permissions denied.

C. Server Implementation

1) Performance: We ran three experiments on a physical machine (Intel(R) Core(TM) i7-2760QM CPU @ 2.40GHz); homomorphic addition, homomorphic addition without using the modulus, and partial decryption. Figure 8 gives the results for each in seconds, for varying bit sizes of N. Addition with the modulus operation is not effected much by the size of N, because the number of bits required remains constant. When homomorphically adding two cipher values without modulus, the bit size of the result value grows proportional to the size of N^2. Therefore the performance decreases as the value grows. The result is an average for the verification group of size 100. Partial decryption, as the results in Figure 7 show, decreases rapidly in performance as N increases.

Looking at these results in a real world scenario, with $size(N) = 2048$-bits, this server can handle ≈ 360 votes per second (excluding other computation overhead). This equates to $1,296,000$ votes per hour, meaning the population of New Zealand would require under four hours for everyone to vote [42]. Given a higher-powered server, or a dedicated hardware server as described in Section VIII, the performance cost of voting in the cloud is not a problem.

2) Security Threats: Like the mobile agents, the biggest threat of the voting model comes from the underlying system and its flaws. This was the primary attack vector used against the Estonian government voting system [7]. An example of a widespread vulnerability discovered in 2014 was Shellshock [43]. This could have rendered a voting system built upon the Unix Bash Shell compromised. Therefore any voting systems must be built using a secure platform, which we will discuss in the next section.

VIII. Specialised System Model

Attacks on voting systems often focus on the underlying software and operating system, not the actual voting scheme [7]. To render these attacks ineffective, a secure voting platform must be developed so it can be easily deployed in the cloud. This could be achieved by either building a secure operating system for voting only, or a hardware platform. In this section we will describe a tally server hardware model which can help solve many problems faced by voting with software based servers. Note that the same approach can be used to include a partial decryption server into the hardware model as well.

Figure 9 below shows a simple model for submitting a vote which could be implemented in a FPGA (Field-Programmable Gate Array). Any data being received must be encrypted with the servers public key. All data (voter decryption keys and group tallies) in storage is also encrypted using another

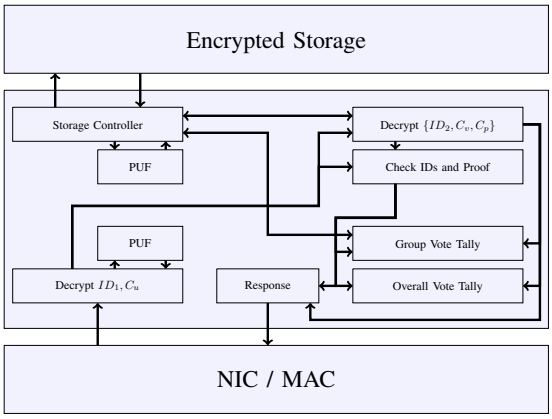

Fig. 9: Hardware Ballot Tally Model.

key pair. It makes use of two PUFs (Physical Unclonable Functions) [44][45], which use the physical properties of the die (i.e. silicon) such that a function can be easily evaluated, but hard to predict. This keeps the private keys secure as no knowledge of them is leaked using conventional attack methods. Both keys for storage can remain private, while the encryption key for transmitting to the server can be made public. For a malicious user to gain access to the private keys, they must have physical access to the server, and run side-channel-attacks. For example in 2013, a group from Berlin Institute of Technology managed to clone a PUF in a laboratory by reading from SRAM (Static RAM) [46].

Another useful feature of developing a hardware solution for keeping a secure vote tally is that is can be easily parallelised. This design can be pipelined so that many votes are being processed at once. For example, as a cipher vote is being decrypted after the packet has been received, another could be at the verification stage, whilst another is being tallied. Therefore, not only does a hardware tally system have better security properties, it can also deliver better performance per unit of power compared to a traditional server [47][48].

IX. Evaluation of Voting Model

We evaluate our system against the requirements described in Section IV. Because of the extra security added by the hardware model, we will use it for this section. We will also assume that side-channel-attacks are not possible because there is no physical access allowed to the server.

1) **Eligibility:** Each eligible voter receives an ID for the ballot. This is sent out via post if the user is enrolled, like paper based systems now. The user submits their verification key to the server, and receives a hash to confirm the key was uploaded correctly. Because we are using the hardware model for the tally server, this process is secure assuming the public key the user received for the server is correct (this can be confirmed at government agents, online and via post). If someone else has submitted a verification key for their ID already, then an error will occur and the user can contact the authorities. The mobile agents (e.g. iOS or Android) must be able to protect the ID and keys saved by the voting application. If malware manages to infect the device and leak this information, the malicious entity can vote on their behalf.
2) **Unreusability:** A voter can only submit one vote during the ballot which can be enforced in the tally servers. This helps protect a user if their mobile agent gets infected. An error will occur if they try and submit a vote but a malicious entity has already voted using their ID. This allows them to contact the authorities. Because the private key of the tally servers are safe, a malicious entity cannot trick the application into thinking they submitted correctly because the hashes will not match.
3) **Untraceability:** Removing a public bulletin board for displaying all cipher votes and IDs helps protect the voters privacy. All individual voters are combined into groups which hides how an individual voted. If a software tally server is used and gets compromised, then each cipher vote and ID could be logged and possibly decrypted after the ballot. Hence we recommend the use of a hardware tally server.
4) **Verifiability:** Using our proposed verification technique described in Section VI-D, a voter can have full confidence that their vote was cast correctly.
5) **Tally Correctness:** Because there are many tally counting servers, all combinations of servers should give the same tally if the ballot was not compromised. Each voting server publishes the number of voters registered, and the number that voted.
6) **Uncoerceability:** The application on the mobile agent should not reveal the cipher vote or the vote value, only if it was successfully included in the final tally or not.
7) **Auditability:** During the verification phase, a voter can test the outcome of dummy votes to see that the result changes correctly.
8) **Accessibility:** By supporting many types of mobile agents, a voter is not restricted by location and can use their existing mobile device to vote. The voting procedure will be handled by the mobile application, where the voter only has to select the ballot and their vote, then cast it. Verification is also done via the tap of a button. Our performance results show that the bit size of N greatly affects the time it takes to cast a vote.
9) **Fairness:** This comes down to implementation. For example a policy can be put in place such that the decryption servers will only accept requests after the ballot has finished.
10) **Soundness:** Voting parameters are encrypted twice, with the servers public key, and user's encryption keys. Then all traffic is secured again using SSL. We have also discussed how to prevent man-in-the-middle attacks and network sniffing.
11) **Integrity:** Many tally and decryption servers are spread over different locations and organisations. This prevents the voting authority from controlling the outcome of the ballot. Currently they can add fake users to the list of IDs, however this problem also exists with paper voting.

12) **Completeness:** All eligible voters receive their ID in the post, and can verify their vote was counted towards the final tally.

X. Evaluation Against Chosen Ciphertext Attacks

Formal definitions of plaintext attacks have been described by Rackoff *et al.* [49], Luby [50] and Bellare *et al.* [51]. We will discuss CCA (Chosen Ciphertext Attacks), in particular CCA1-security (security against non-adaptive chosen ciphertext attacks) because CCA2-security (security against adaptive chosen ciphertext attacks) does not allow for homomorphic operations. CCA1-security can be used instead of CPA-security (security against chosen plaintext attacks) for e-voting [52].

A CCA attack is where an entity has access to the decryption oracle, and can request the plaintext of arbitrary nonsense ciphertext values to try and recover the private key [49][53]. In our proposed scheme, any voter has access to the decryption oracle in order to verify that their vote has been included in the final tally. The decryption servers should only accept requests from authenticated voters after the ballot. Because the number of verifications a voter must perform to verify their vote is known (i.e. how many groups need to be checked to verify the final tally), each voter should have a limitation on the number of decryption requests they can perform.

Even with these limitations in place, if a user still manages to submit enough decryption requests to recover the private key, they gain no extra knowledge of the ballot or how individual users voted. This is because cipher votes are not stored on the voting servers or included on a public bulletin board, instead they are only added to the group's cipher tally then discarded. By using a hardware server with extra private key protection as described in Section VIII, man-in-the-middle attacks cannot reveal cipher votes. During verification, the decryption servers receive a valid and dummy group cipher value, therefore even with knowledge of the private key, it would only reveal that a user voted for "yes" or "no", but not which one. The only case where this is not true is if everyone in the group voted "yes" or everyone voted "no". However because a user is not directly linked to an ID, the malicious user still does not know how individual voters voted. The mobile application also prevents the cipher votes from been seen, as the application only reveals ballot information, and whether or not the user's vote was successfully cast. There does however need to be more research into how to stop the cipher vote been leaked by malware.

XI. Current Problems and Future Work

There are a few problems that need to be addressed for a system such as the one we propose to be at least as secure as paper voting.

- **Mobile Agents:** There needs to be more work into guaranteeing that any information (keys and IDs) stored in the voting application remains secure and cannot be leaked via malware. Also the performance of encryption and proof generation needs to be improved, possibly by writing more optimised code. This may include a list of verified/certified mobile devices.
- **Ballot Key Generation:** Currently the key generation for the ballot has to been done by a single entity before being distributed to the tally and decryption servers. This is a security threat, and can be addressed by generating the key using distributed computation [31] similar to how the decryption works.
- **Corrupt Vote:** Removing a vote once cast is currently impossible with our design because individuals votes are not saved, only the group tallies. If a corrupt vote has been cast, a technique needs to be designed so that a group can revote.
- **Hardware Server:** Given that a hardware server can provide much better security, we would like to design a working version and test the security in order to prove our claims.
- **Panic Voting:** We would like to address the issue of a voter being forced (e.g. by gunpoint) into voting a particular way. One possible solution is to have a panic mode where the vote appears to have been cast correctly [54].

XII. Conclusion

This paper presents a scheme for a partially homomorphic cloud-based mobile voting system, with implementation results to show its practicality. We have addressed some of the current issues faced by e-voting, by improving voter privacy, and proposing the use of a hardware tally server for better security. Our scheme has been evaluated against a list of proposed requirements, which any e-voting system should meet. Currently the main limitation of such a scheme is the mobile agents, which lack security guarantees that a government voting system requires. Once addressed, like other future work, this remote e-voting scheme should be as secure as current paper voting techniques.

XIII. Acknowledgements

This research is supported by STRATUS (Security Technologies Returning Accountability, Trust and User-Centric Services in the Cloud) (https://stratus.org.nz), a science investment project funded by the New Zealand Ministry of Business, Innovation and Employment (MBIE). The authors would also like to acknowledge Raja Naeem Akram for his contributions to the mobile voting requirements list.

References

[1] Dates and turnout of New Zealand General Elections. [Online] http://www.elections.org.nz/events/past-events/general-elections-1853-2014-dates-and-turnout (Accessed 08/06/15).

[2] Tim Storer and Ishbel Duncan. Electronic Voting in the UK: Current Trends in Deployment, Requirements and Technologies. In *PST*, 2005.

[3] Jan Gerlach and Urs Gasser. Three Case Studies from Switzerland: E-voting. *Berkman Center Research Publication No*, 3, 2009.

[4] Voter and Non-Voter Satisfaction Survey 2011. [Online] http://www.elections.org.nz/events/past-events-0/2011-general-election/reports-and-surveys-2011-general-election/voter-and-non (Accessed 08/06/15).

[5] Ryan K. L. Ko. Cloud Computing in Plain English. *Crossroads*, 16(3):5–6, March 2010.

[6] R Michael Alvarez, Thad E Hall, and Alexander H Trechsel. Internet Voting in Comparative Perspective: The Case of Estonia. *PS: Political Science & Politics*, 42(03):497–505, 2009.

[7] Drew Springall, Travis Finkenauer, Zakir Durumeric, Jason Kitcat, Harri Hursti, Margaret MacAlpine, and J Alex Halderman. Security Analysis of the Estonian Internet Voting System. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 703–715, 2014.

[8] Mark A. Will and Ryan K.L. Ko. A guide to homomorphic encryption. In *The Cloud Security Ecosystem: Technical, Legal, Business and Management Issues*, pages 101–127. Elsevier, 2015.

[9] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in cryptology—EUROCRYPT'99*, pages 223–238. Springer, 1999.

[10] Ivan Damgård and Mads Jurik. A generalisation, a simpli. cation and some applications of paillier's probabilistic public-key system. In *Public Key Cryptography*, pages 119–136. Springer, 2001.

[11] Ben Adida. Helios: Web-based open-audit voting. In *USENIX Security Symposium*, volume 17, pages 335–348, 2008.

[12] Josh Benaloh. Simple verifiable elections. In *Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop 2006 on Electronic Voting Technology Workshop*, pages 5–5. USENIX Association, 2006.

[13] Kazue Sako and Joe Kilian. Receipt-free mix-type voting scheme. In *Advances in Cryptology—EUROCRYPT'95*, pages 393–403. Springer, 1995.

[14] Feng Hao and Matthew Nicolas Kreeger. *Every Vote Counts: Ensuring Integrity in Large-Scale DRE-based Electronic Voting.* Newcastle University, Computing Science, 2011.

[15] Ronald Cramer, Matthew Franklin, Berry Schoenmakers, and Moti Yung. Multi-authority secret-ballot elections with linear work. In *Advances in Cryptology—EUROCRYPT'96*, pages 72–83. Springer, 1996.

[16] Byoungcheon Lee and Kwangjo Kim. Receipt-free Electronic Voting through Collaboration of Voter and Honest Verifier. In *Proceeding of JW-ISC2000*, pages 101–108, 2000.

[17] Olivier Baudron, Pierre-Alain Fouque, David Pointcheval, Jacques Stern, and Guillaume Poupard. Practical Multi-Candidate Election System. In *Proceedings of the Twentieth Annual ACM Symposium on Principles of Distributed Computing*, pages 274–283. ACM, 2001.

[18] Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-Resistant Electronic Elections. In *Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society*, pages 61–70. ACM, 2005.

[19] Josh D Cohen and Michael J Fischer. A Robust and Verifiable Cryptographically Secure Election Scheme. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 372–382. IEEE, 1985.

[20] Feng Hao, Brian Randell, and Dylan Clarke. *Self-Enforcing Electronic Voting*. Springer, 2012.

[21] Dylan Clarke, Feng Hao, and Brian Randell. Analysis of Issues and Challenges of E-Voting in the UK. In *Security Protocols XX*, pages 126–135. Springer, 2012.

[22] Arne Ansper, Sven Heiberg, Helger Lipmaa, Tom André Øverland, and Filip Van Laenen. Security and Trust for the Norwegian E-voting Pilot Project E-valg 2011. In *Identity and Privacy in the Internet Age*, pages 207–222. Springer, 2009.

[23] Reto E Koenig, Philipp Locher, and Rolf Haenni. Attacking the Verification Code Mechanism in the Norwegian Internet Voting System. In *E-Voting and Identify*, pages 76–92. Springer, 2013.

[24] Scott Wolchok, Eric Wustrow, Dawn Isabel, and J Alex Halderman. Attacking the Washington, DC Internet Voting System. In *Financial Cryptography and Data Security*, pages 114–128. Springer, 2012.

[25] Nicole J Goodman. Internet Voting in a Local Election in Canada. In *The Internet and Democracy in Global Perspective*, pages 7–24. Springer, 2014.

[26] Dimitrios Zissis and Dimitrios Lekkas. Securing e-government and e-voting with an open cloud computing architecture. *Government Information Quarterly*, 28(2):239–251, 2011.

[27] Chun-I Fan and Wei-Zhe Sun. An Efficient Multi-Receipt Mechanism for Uncoercible Anonymous Electronic Voting. *Mathematical and Computer Modelling*, 48(9):1611–1627, 2008.

[28] Dimitris A Gritzalis. Principles and Requirements for a Secure e-voting System. *Computers & Security*, 21(6):539–556, 2002.

[29] Wei-Chi Ku and Sheng-De Wang. A Secure and Practical Electronic Voting Scheme. *Computer Communications*, 22(3):279–286, 1999.

[30] Horng-Twu Liaw. A Secure Electronic Voting Protocol for General Elections. *Computers & Security*, 23(2):107–119, 2004.

[31] Ronald Cramer, Ivan Damgard, and Jesper Buus Nielsen. Secure multiparty computation and secret sharing-an information theoretic appoach. *Book Draft*, 2012.

[32] libhcs. [Online] https://github.com/Tiehuis/libhcs (Accessed 28/05/15).

[33] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Advances in Cryptology*, pages 10–18. Springer, 1985.

[34] The GNU Multiple Precision Arithmetic Library. [Online] https://gmplib.org (Accessed 28/05/15).

[35] ScientiaMobile. Mobile Overview Report October - December 2014. [Online] http://www.scientiamobile.com/page/wp-content/uploads/2015/02/MOVR_Q4_2014_v6.pdf (Accessed 07/06/15).

[36] iOS Developer Library - About App Distribution. [Online] https://developer.apple.com/library/ios/documentation/IDEs/Conceptual/AppDistributionGuide/Introduction/Introduction.html (Accessed 07/06/15).

[37] Android - Signing Your Applications. [Online] http://developer.android.com/tools/publishing/app-signing.html (Accessed 07/06/15).

[38] Alexios Mylonas, Stelios Dritsas, Bill Tsoumas, and Dimitris Gritzalis. Smartphone security evaluation-the malware attack case. *SECRYPT*, 11:25–36, 2011.

[39] Adrienne Porter Felt, Matthew Finifter, Erika Chin, Steve Hanna, and David Wagner. A survey of mobile malware in the wild. In *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, pages 3–14. ACM, 2011.

[40] Yajin Zhou and Xuxian Jiang. Dissecting android malware: Characterization and evolution. In *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 95–109. IEEE, 2012.

[41] Alexios Mylonas, Marianthi Theoharidou, and Dimitris Gritzalis. Assessing privacy risks in android: a user-centric approach. In *Risk Assessment and Risk-Driven Testing*, pages 21–37. Springer, 2014.

[42] New Zealand Population Clock. [Online] http://www.stats.govt.nz/tools_and_services/population_clock.aspx (Accessed 07/06/15).

[43] John Graham-Cumming. Inside Shellshock: How hackers are using it to exploit systems. [Online] https://blog.cloudflare.com/inside-shellshock/ (Accessed 08/06/15), September 2014.

[44] G Edward Suh and Srinivas Devadas. Physical unclonable functions for device authentication and secret key generation. In *Proceedings of the 44th annual Design Automation Conference*, pages 9–14. ACM, 2007.

[45] Jorge Guajardo, Sandeep S Kumar, G-J Schrijen, and Pim Tuyls. Physical unclonable functions and public-key crypto for FPGA IP protection. In *Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on*, pages 189–195. IEEE, 2007.

[46] Clemens Helfmeier, Christian Boit, Dmitry Nedospasov, and J-P Seifert. Cloning physically unclonable functions. In *Hardware-Oriented Security and Trust (HOST), 2013 IEEE International Symposium on*, pages 1–6. IEEE, 2013.

[47] Ying Liu, Khaled Benkrid, AbdSamad Benkrid, and Server Kasap. An fpga-based web server for high performance biological sequence alignment. In *Adaptive Hardware and Systems, 2009. AHS 2009. NASA/ESA Conference on*, pages 361–368. IEEE, 2009.

[48] Mark A. Will. *Real-time Image Processing*. Honours thesis, The University of Waikato, 2013, [Online] http://markwill.me/publications/ (Accessed 07/06/15).

[49] Charles Rackoff and Daniel R Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In *Advances in Cryptology—CRYPTO'91*, pages 433–444. Springer, 1992.

[50] Michael George Luby. *Pseudorandomness and cryptographic applications*. Princeton University Press, 1996.

[51] Mihir Bellare, Anand Desai, Eron Jokipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption. In *Foundations of Computer Science, 1997. Proceedings., 38th Annual Symposium on*, pages 394–403. IEEE, 1997.

[52] Helger Lipmaa. On the cca1-security of elgamal and damgård's elgamal. In *Information Security and Cryptology*, pages 18–35. Springer, 2011.

[53] Pierre-Alain Fouque and David Pointcheval. Threshold cryptosystems secure against chosen-ciphertext attacks. In *Advances in Cryptology—ASIACRYPT 2001*, pages 351–368. Springer, 2001.

[54] Jeremy Clark and Urs Hengartner. Selections: Internet voting with over-the-shoulder coercion-resistance. In *Financial Cryptography and Data Security*, pages 47–61. Springer, 2012.