TallyGuard: Privacy Preserving Tallied-as-cast Guarantee

Athish Pranav Dharmalingam[1 [0009−0000−7326−4662]], Sai Venkata Krishnan[1 [0009−0008−9609−772X]], KC Sivaramakrishnan[1 [0000−0002−3491−1780]], and N.S. Narayanaswamy[1 [0000−0002−8771−3921]]

Indian Institute of Technology, Madras, Chennai, India

Abstract. This paper presents a novel approach to verifiable vote tallying using additive homomorphism, which can be appended to existing voting systems without modifying the underlying infrastructure. Existing End-to-End Verifiable (E2E-V) systems like Belenios and ElectionGuard rely on distributed trust models or are vulnerable to decryption compromises, making them less suitable for general elections. Our approach introduces a tamper-evident commitment to votes through cryptographic hashes derived from homomorphic encryption schemes such as Paillier. The proposed system provides tallied-as-cast verifiability without revealing individual votes, thereby preventing coercion. The system also provides the ability to perform public verification of results. We also show that this system can be transitioned to quantum-secure encryption like Regev for future-proofing the system. We discuss how to deploy this system in a real-world scenario, including for general political elections, analyzing the security implications and report on the limitations of this system. We believe that the proposed system offers a practical solution to the problem of verifiable vote tallying in general elections.

Keywords: Electronic voting machine · Voter-verified paper audit trail · Tallied-as-cast · Homomorphic encryption

1 Introduction

In recent years, the integrity and transparency of elections have become critical concerns, especially in general elections. Traditional voting methods like paper based voting, while reliable, have issues with scalability, security and efficiency. This has led to the exploration of electronic and online voting systems [4,7,5,9], which aim to address these concerns, but also introduce new security challenges [8,24].

We consider the setting of an offline, polling booth, electronic voting system for general elections. Assume that an independent election commission runs the elections at the constituency level. The election commission sets up the elections, the polling booths, and runs the counting process. The votes are recorded both electronically and in a secure printed slip of paper that the voter is able to see but not take home. The printed slip is collected and stored securely alongside

the electronic vote. The electronic votes are tallied to determine the election outcome. A few of the electronic voting machines are selected and their votes are tallied with the printed slips to ensure the integrity of the election. Election observers are allowed to witness the voting and counting process to ensure that the election is conducted fairly.

This system has the advantage that the association between the voter and their votes end at the point of voting, ensuring voter privacy and coercion resistance. The printed vote preference gives the voter the chance to confirm that their votes have been cast correctly. However, given the votes are tallied electronically, and only a small percentage of electronic votes are tallied against the printed votes, there is no guarantee that the votes are tallied correctly.

How can we ensure that the electronic votes are tallied correctly in such a system? We would like to do this as an additional verification step, without affecting the existing voting process. In particular, we would like any member of the public to be able to (a) validate that their vote has been recorded as intended and (b) all the collected votes are tallied correctly to determine the outcome of the election. At the same time, we would also like to ensure that an individual voter's choice is not revealed to anyone, intentionally or otherwise, to prevent coercion.

We can summarize our requirements as follows:

1. **Cast-as-Intended:** voters must be able to get convincing evidence that their votes have been recorded as intended.
2. **Recorded-as-Cast:** voters should be able to verify that their vote has been recorded correctly.
3. **Coercion Resistance:** voters must not be able to prove to anyone how they voted.
4. **Voter Privacy:** no authority or a group of authorities should be able to link a voter to their vote.
5. **Tallied-as-Cast:** any member of the public should be able to verify that all the recorded votes have been tallied correctly.
6. **Deployability:** the system should be deployable in the existing voting system as an addition, without affecting the existing process.

We propose TallyGuard, a novel approach to verifying vote tallying that can be appended to an existing offline electronic voting system. Like existing end-to-end verifiable voting systems such as Helios [1], Belenios [7], Electionguard [4], StarVote [2], we use additive homomorphic encryption to tally the votes. The voters are also given a hash of their vote as an acknowledgment, which they can use to verify that their vote has been recorded correctly in a public list of cast vote hashes. The key novelty of our system is that unlike existing approaches systems, TallyGuard does not require decrypting the encrypted votes in the entire election process. The public list of cast votes is used to verify that the election result is valid. In the rest of the paper, we present the approach formally and then discuss the deployment approach.

2 Tallied-as-cast with Paillier encryption

In this section, we describe how TallyGuard uses a novel variant of the Paillier encryption to guarantee privacy preserving tallied-as-cast property. Paillier encryption [19] is a well-known public key cryptosystem, which is based on the decisional composite residuosity problem (DCRA). It supports additive homomorphism. We describe the Paillier cryptosystem below.

Key generation: 2 large prime numbers p and q are chosen randomly such that $\gcd(pq, (p-1)(q-1)) = 1$. Randomly select a base g such that $g \in \mathbb{Z}^*_{n^2}$, where $n = pq$. The public key k_{pub} is $\{g, n\}$. Compute $\lambda = \text{lcm}(p-1, q-1)$ and $\mu = L(g^\lambda \mod n^2)^{-1} \mod n$, where $L(x) = \frac{x-1}{n}$. The private key k_{priv} is $\{\lambda, \mu\}$.

Encryption: The encryption function is $\epsilon(m, k_{pub} = \{\text{g,n}\}) = g^m \cdot r^n \mod n^2$ where r is a random nonce such that $0 < r < n$ and $\gcd(r, n) = 1$ which is generated when the function is invoked.

Decryption: Let the ciphertext be c. The decryption function is $decrypt(c, k_{priv} = \{\lambda, \mu\}) = L(c^\lambda \mod n^2) \cdot \mu \mod n$.

TallyGuard is configured to use Paillier encryption with the following 2 changes:

1. TallyGuard does not use the private key k_{priv}. Hence, λ and μ are safely discarded after key generation.
2. The random nonce r is generated and passed as input to the encryption function.

The direct implication of the first change is that one cannot decrypt the encrypted votes. Hence, for clarity, we call an encrypted vote as a *vote hash*. The vote hashes still have the additive homomorphism property.

Notation. One dimensional vector is represented as $\overrightarrow{\boldsymbol{V}}$ and the two dimensional vector/matrix is represented as $\boldsymbol{H}$. Scalars are represented in normal font.

2.1 Pre-election setup

As part of the pre-election setup phase, the election commission resets the voting machine, and loads the candidate list for the upcoming election. At this point, the election commission generates a k_{pub} and k_{priv} pair, and immediately discards the private key k_{priv}. We assume that no stakeholder has access to k_{priv}. We also make the reasonable assumption that the deriving the k_{priv} from the k_{pub} or any other information available in TallyGuard is computationally infeasible. Then, k_{pub} is loaded into the voting machine. The election commission also releases the k_{pub} to the public before the election takes place, and thus

committing to the public key. For the sake of the discussion, we assume that the tallied-as-cast verification is performed at the level of individual voting machine. This will produce tamper evidence at the level of a voting machine. If the same public key is used for all the voting machines in a given election, then the tamper evidence is at the level of the constituency.

Note: For a election with N voters, the public key parameter n must be chosen such that $n > N$.

2.2 Recording the vote

We represent the vote of v^{th} voter in form of a bit vector $\overrightarrow{\boldsymbol{V_v}}$ of dimensions l, where l is the number of candidates. When a voter casts his vote for j^{th} candidate, $V_{v,j} = 1$ and others are set to 0. For hashing the vote, we treat each bit of the vector as an individual plain text and hash it using the Paillier encryption scheme. The vote hash for the v^{th} voter is represented as $\overrightarrow{\boldsymbol{H_v}}$.

$$\overrightarrow{\boldsymbol{V}}_{\boldsymbol{v}} = \begin{bmatrix} V_{v,1} \\ V_{v,2} \\ \vdots \\ V_{v,l} \end{bmatrix}, \overrightarrow{\boldsymbol{H}}_{\boldsymbol{v}} = \begin{bmatrix} H_{v,1} \\ H_{v,2} \\ \vdots \\ H_{v,l} \end{bmatrix} = \begin{bmatrix} \epsilon(V_{v,1}, k_{pub}, r_{v,1}) \\ \epsilon(V_{v,2}, k_{pub}, r_{v,2}) \\ \vdots \\ \epsilon(V_{v,l}, k_{pub}, r_{v,l}) \end{bmatrix}$$

Independent random nonces $r_{v,j}$ are generated for each bit of the vote and passed as input to the corresponding encryption function. The aggregate nonce vector $\overrightarrow{\boldsymbol{R_v}}$ is the product of all the random nonces used and the result vector $\overrightarrow{\boldsymbol{W_v}}$ is sum of votes that have been cast, where v is the number votes recorded from start of polling.

$$\overrightarrow{\boldsymbol{R}}_{\boldsymbol{v}} = \begin{bmatrix} R_{v,1} \\ R_{v,2} \\ \vdots \\ R_{v,l} \end{bmatrix} = \begin{bmatrix} \prod_{i=1}^{v} r_{i,1} \\ \prod_{i=1}^{v} r_{i,2} \\ \vdots \\ \prod_{i=1}^{v} r_{i,l} \end{bmatrix}, \overrightarrow{\boldsymbol{W}}_{\boldsymbol{v}} = \begin{bmatrix} W_{v,1} \\ W_{v,2} \\ \vdots \\ W_{v,l} \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{v} V_{i,1} \\ \sum_{i=1}^{v} V_{i,2} \\ \vdots \\ \sum_{i=1}^{v} V_{i,l} \end{bmatrix}$$

The vote hash acts as the evidence for the vote cast by the voter. A copy of this hash is given to the voter as a receipt. The voting machine stores the result vector $\overrightarrow{\boldsymbol{W}}_{\boldsymbol{v}}$, the aggregate nonce vector $\overrightarrow{\boldsymbol{R}}_{\boldsymbol{v}}$, the vote hash $\overrightarrow{\boldsymbol{H}}_{\boldsymbol{v}}$ and drops the vote $\overrightarrow{\boldsymbol{V}}_{\boldsymbol{v}}$.

2.3 Post-election verification

Recall that the public key k_{pub} had been made public during the pre-election setup. Let the total number of votes cast be N. The election commission publishes the election result $\overrightarrow{\boldsymbol{W}}_{\boldsymbol{N}}$, which is the sum of all the votes cast. In addition,

the election commission also publishes the aggregate nonce vector $\overrightarrow{\boldsymbol{R}_{\boldsymbol{N}}}$ and the individual vote hashes $\overrightarrow{\boldsymbol{H}}$ in Bulletin Board (BB).

An individual voter can verify that his vote has been correctly recorded by checking that their vote hash is included in the BB. Any member of the public can verify the correctness of the election result based on the published vote hashes, public key and the aggregate nonce vector as follows:

Theorem 1 (Tallied-as-cast verification-Paillier).

$$\begin{bmatrix} \prod_{i=1}^{N} H_{i,1} \\ \prod_{i=1}^{N} H_{i,2} \\ \vdots \\ \prod_{i=1}^{N} H_{i,l} \end{bmatrix} = \begin{bmatrix} \epsilon(W_{N,1}, k_{pub}, R_{N,1}) \\ \epsilon(W_{N,2}, k_{pub}, R_{N,2}) \\ \vdots \\ \epsilon(W_{N,l}, k_{pub}, R_{N,l}) \end{bmatrix}$$

The key difference between TallyGuard and other schemes is that we do not decrypt the homomorphically computed hash to obtain the result. Decrypting individual votes is not possible as the private key has been discarded during the pre-election setup. Instead, we verify the tallied-as-cast by validating the hash of the result with the published vote hashes.

TallyGuard offers flexibility for the granularity of the verification process. If different public keys are used for different voting machines used in an election, then the verification process can be done at the voting machine level (at the cost of releasing the vote count for each voting machine). However, an election for a constituency typically involves multiple voting machines. In such cases, the tallied-as-cast verification can be done at the constituency level by using the same public key for all the voting machines in the constituency. The aggregate nonce vector is the product of the aggregate nonce vectors from all the voting machines in the constituency. In the case of constituency level verification, result is only required to be published at the constituency level, which is the result of the election in that constituency.

2.4 Correctness of verification

The proof of Theorem 1 is as follows. Additive homomorphism property of Paillier encryption ensures that the product of the vote hashes is the hash of the sum of the votes with the product of the random nonces.

Additive homomorphism

$$\begin{aligned}
&\epsilon(m_1, k_{pub}, r_1) \cdot \epsilon(m_2, k_{pub}, r_2) \mod n^2 \\
&\quad = (g^{m_1} \cdot r_1^n \mod n^2) \cdot (g^{m_2} \cdot r_2^n \mod n^2) \\
&\quad = g^{m_1+m_2} \cdot (r_1 \cdot r_2)^n \mod n^2 \\
&\quad = \epsilon(m_1 + m_2, k_{pub}, r_1 \cdot r_2)
\end{aligned}$$

$$\begin{bmatrix} \prod_{i=1}^{N} H_{i,1} \\ \prod_{i=1}^{N} H_{i,2} \\ \vdots \\ \prod_{i=1}^{N} H_{i,l} \end{bmatrix} = \begin{bmatrix} \epsilon(\sum_{i=1}^{N} V_{i,1}, k_{pub}, \prod_{i=1}^{N} r_{i,1}) \\ \epsilon(\sum_{i=1}^{N} V_{i,2}, k_{pub}, \prod_{i=1}^{N} r_{i,2}) \\ \vdots \\ \epsilon(\sum_{i=1}^{N} V_{i,l}, k_{pub}, \prod_{i=1}^{N} r_{i,l}) \end{bmatrix} = \begin{bmatrix} \epsilon(W_{N,1}, k_{pub}, R_{N,1}) \\ \epsilon(W_{N,2}, k_{pub}, R_{N,2}) \\ \vdots \\ \epsilon(W_{N,l}, k_{pub}, R_{N,l}) \end{bmatrix}$$

3 Quantum secure TallyGuard

The Paillier cryptosystem is increasingly considered unsuitable in light of recent advancements in quantum computing, which have heightened the risks associated with cryptographic schemes reliant on decisional composite residuosity problem (DCRA), discrete log problem (DLP) and RSA problem. To ensure long-term security, it is essential to adopt cryptographic primitives that are resilient to quantum attacks. The Regev encryption scheme [20] is one such quantum resistant approach that enables homomorphic additions, making it a valuable candidate for secure computations. A variant of this scheme [25] is particularly well-suited to our use case, as it extends support to general addition operations, addressing the limitations of the original scheme proposed, which was restricted to bitwise addition. We describe the modified Regev encryption scheme below.

Key generation: The security parameter n is chosen which denotes the dimension of the lattice used. χ is binomial distribution on $\mathbb{Z}_2$. Modulus q is a prime integer. Message space is t such that $t < q$. Choose $\overrightarrow{\boldsymbol{s}}' \leftarrow \mathbb{Z}_{\pm q}^n$ uniformly and randomly such that $\overrightarrow{\boldsymbol{s}} \leftarrow (1, \overrightarrow{\boldsymbol{s}}')$. Choose n' such that $n' < \lfloor \frac{q}{2t} \rfloor$. Choose randomly $\boldsymbol{A'} \leftarrow \mathbb{Z}_q^{n' \times n}$. Choose $\overrightarrow{\boldsymbol{e}} \leftarrow \chi^n$. Calculate $\overrightarrow{\boldsymbol{b}} = \boldsymbol{A'}\overrightarrow{\boldsymbol{s}}' + \overrightarrow{\boldsymbol{e}} \in \mathbb{Z}_q^{n'}$. The public key k_{pub} is $\boldsymbol{A} = [\overrightarrow{\boldsymbol{b}}, -\boldsymbol{A'}] \in \mathbb{Z}_q^{n' \times (n+1)}$. The private key k_{priv} is $\overrightarrow{\boldsymbol{s}}$.

Encryption: The encryption function is $\overrightarrow{\boldsymbol{\epsilon}}(m, k_{pub}) = \boldsymbol{A}^T \times \overrightarrow{\boldsymbol{r}} + \overrightarrow{\boldsymbol{m}} \cdot \lfloor \frac{q}{2t} \rfloor$ where $\boldsymbol{A}$ is public key, $\overrightarrow{\boldsymbol{m}} \leftarrow [m, 0, 0, \ldots]$ and $\overrightarrow{\boldsymbol{r}} \in (0,1)^{n'}$.

Decryption: Let the ciphertext be $\overrightarrow{\boldsymbol{c}}$. The decryption function is $decrypt(c, k_{priv}) = \lfloor \frac{1}{\lfloor \frac{q}{2t} \rfloor} ((\overrightarrow{\boldsymbol{c}}^{\boldsymbol{T}} \cdot \overrightarrow{\boldsymbol{s}}) \mod q) \rfloor \mod t$.

Similar to TallyGuard's use of Paillier, TallyGuard does not use the private key, and hence is discarded after key generation and the random vector $\overrightarrow{r}$ is taken as input to the encryption function.

3.1 Pre-election setup

The pre-election setup is similar to the Paillier case, where the public key is loaded into the voting machines and is made public. For sake of simplicity we assume the verification is done at the voting machine level.

Note: For a election with N voters, Regev encryption scheme will hold only when $N < t < q$ and $n' < \frac{1}{N}\lfloor\frac{q}{2t}\rfloor$.

3.2 Recording the vote

The vote representation remains the same as the Paillier case. We use Regev encryption here to hash the vote.

$$\boldsymbol{H_v} = \begin{bmatrix} \overrightarrow{\boldsymbol{H}}_{\boldsymbol{v,1}} \\ \overrightarrow{\boldsymbol{H}}_{\boldsymbol{v,2}} \\ \vdots \\ \overrightarrow{\boldsymbol{H}}_{\boldsymbol{v,l}} \end{bmatrix} = \begin{bmatrix} \overrightarrow{\epsilon}(V_{v,1}, k_{pub}, \overrightarrow{\boldsymbol{r}}_{\boldsymbol{v,1}}) \\ \overrightarrow{\epsilon}(V_{v,2}, k_{pub}, \overrightarrow{\boldsymbol{r}}_{\boldsymbol{v,2}}) \\ \vdots \\ \overrightarrow{\epsilon}(V_{v,l}, k_{pub}, \overrightarrow{\boldsymbol{r}}_{\boldsymbol{v,l}}) \end{bmatrix}$$

Like before, we store the aggregate value of the random numbers used in the encryption. Here the aggregate is the vector sum of $\overrightarrow{r}$ used in the encryption. When v votes have been polled, the aggregate $\boldsymbol{R_v}$ is:

$$\boldsymbol{R_v} = \begin{bmatrix} \overrightarrow{\boldsymbol{R}}_{\boldsymbol{v,1}} \\ \overrightarrow{\boldsymbol{R}}_{\boldsymbol{v,2}} \\ \vdots \\ \overrightarrow{\boldsymbol{R}}_{\boldsymbol{v,l}} \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{v} \overrightarrow{\boldsymbol{r}}_{\boldsymbol{i,1}} \\ \sum_{i=1}^{v} \overrightarrow{\boldsymbol{r}}_{\boldsymbol{i,2}} \\ \vdots \\ \sum_{i=1}^{v} \overrightarrow{\boldsymbol{r}}_{\boldsymbol{i,l}} \end{bmatrix}$$

3.3 Post election verification

The result verification process is very similar to Paillier case, with the only difference being the dimension of the vote hashes and aggregate information. Following theorem is used to verify the result:

Theorem 2 (Tallied-as-cast verification-Regev).

$$\sum_{i=1}^{N} \boldsymbol{H_i} = \begin{bmatrix} \overrightarrow{\epsilon}(W_{N,1}, k_{pub}, \overrightarrow{\boldsymbol{R}}_{\boldsymbol{v,1}}) \\ \overrightarrow{\epsilon}(W_{N,2}, k_{pub}, \overrightarrow{\boldsymbol{R}}_{\boldsymbol{v,2}}) \\ \vdots \\ \overrightarrow{\epsilon}(W_{N,l}, k_{pub}, \overrightarrow{\boldsymbol{R}}_{\boldsymbol{v,l}}) \end{bmatrix}$$

The proof of Theorem 2 is similar Paillier-based solution (Theorem 1). To verify the correctness of the theorem we use the additive homomorphism property of the Regev encryption scheme.

Additive homomorphisim

$$\begin{aligned} \overrightarrow{\epsilon}(m_1, k_{pub}, \overrightarrow{\boldsymbol{r}}_{\boldsymbol{1}}) + \overrightarrow{\epsilon}(m_2, k_{pub}, \overrightarrow{\boldsymbol{r}}_{\boldsymbol{2}}) &= (A^T \times \overrightarrow{\boldsymbol{r}}_{\boldsymbol{1}} + \overrightarrow{\boldsymbol{m}}_{\boldsymbol{1}} \cdot \lfloor \frac{q}{2t} \rfloor) \\ &\quad + (A^T \times \overrightarrow{\boldsymbol{r}}_{\boldsymbol{2}} + \overrightarrow{\boldsymbol{m}}_{\boldsymbol{2}} \cdot \lfloor \frac{q}{2t} \rfloor) \\ &= A^T \times (\overrightarrow{\boldsymbol{r}}_{\boldsymbol{1}} + \overrightarrow{\boldsymbol{r}}_{\boldsymbol{2}}) + (\overrightarrow{\boldsymbol{m}}_{\boldsymbol{1}} + \overrightarrow{\boldsymbol{m}}_{\boldsymbol{2}}) \cdot \lfloor \frac{q}{2t} \rfloor \\ &= \overrightarrow{\epsilon}(m_1 + m_2, k_{pub}, \overrightarrow{\boldsymbol{r}}_{\boldsymbol{1}} + \overrightarrow{\boldsymbol{r}}_{\boldsymbol{2}}) \end{aligned}$$

The proof of the theorem is as follows:

$$\begin{aligned} \sum_{i=1}^{N} \boldsymbol{H_i} &= \begin{bmatrix} \sum_{i=1}^{N} \overrightarrow{\boldsymbol{H}}_{\boldsymbol{i,1}} \\ \sum_{i=1}^{N} \overrightarrow{\boldsymbol{H}}_{\boldsymbol{i,2}} \\ \vdots \\ \sum_{i=1}^{N} \overrightarrow{\boldsymbol{H}}_{\boldsymbol{i,l}} \end{bmatrix} = \begin{bmatrix} \overrightarrow{\epsilon}(\sum_{i=1}^{N} V_{i,1}, k_{pub}, \sum_{i=1}^{N} \overrightarrow{\boldsymbol{r}}_{\boldsymbol{i,1}}) \\ \overrightarrow{\epsilon}(\sum_{i=1}^{N} V_{i,2}, k_{pub}, \sum_{i=1}^{N} \overrightarrow{\boldsymbol{r}}_{\boldsymbol{i,2}}) \\ \vdots \\ \overrightarrow{\epsilon}(\sum_{i=1}^{N} V_{i,l}, k_{pub}, \sum_{i=1}^{N} \overrightarrow{\boldsymbol{r}}_{\boldsymbol{i,l}}) \end{bmatrix} \\ &= \begin{bmatrix} \overrightarrow{\epsilon}(W_{N,1}, k_{pub}, \overrightarrow{\boldsymbol{R}}_{\boldsymbol{N,1}}) \\ \overrightarrow{\epsilon}(W_{N,2}, k_{pub}, \overrightarrow{\boldsymbol{R}}_{\boldsymbol{N,2}}) \\ \vdots \\ \overrightarrow{\epsilon}(W_{N,l}, k_{pub}, \overrightarrow{\boldsymbol{R}}_{\boldsymbol{N,l}}) \end{bmatrix} \end{aligned}$$

4 Deployment in Indian general elections

As an illustration, we consider the deployment of TallyGuard in Indian general elections. The Indian general elections is the largest democratic elections in the world. In the Lok Sabha, the lower house of India's bicameral parliament, there are 543 constituencies with the eligible voter population of 968 million [6]. The main form of voting is Direct-Recording Electronic (DRE) voting using an Electronic Voting Machine (EVM), along with postal voting for exceptional cases.

4.1 EVM-based voting in India

The EVM consists of two units – Control Unit (CU) and the Ballot Unit (BU). The BU facilitates the cast of individual votes which contains a ballot paper containing candidates and buttons corresponding to each candidate. The CU is responsible to store the votes and controls the BU's operations. The system is powered by a 7.5 V battery inserted into the control unit specially to facilitate voting in regions that lack access to electricity. The EVMs are completely isolated from outside environment in electricity and communication. The votes recorded are stored in the EEPROM present in the CU. A single machine can record a maximum of 2000 votes [23].

The EVMs used in India are equipped with a Voter Verified Paper Audit Trail (VVPAT). When the voter casts the vote in an EVM, a printed slip appears through a sealed window for a few seconds. The printed slip contains the name and party affiliation of the candidate for whom the vote was cast. The VVPAT slip acts as a confirmation for the voter to identify if their vote is cast-as-intended. During the result compilation, 2% of EVM– VVPAT [23] pairs are chosen per constituency and the VVPAT slips are counted to cross verify the counts shown in the EVM. This is carried out as an risk limiting audit to ensure the correctness of the result partially.

On the polling day, a pre-election audit is performed at the polling station before the polls are open. During the pre-election audit, the polling officers and observers belonging to the different political parties, cast a small number of votes in full public view. After the mock polls are over, the results are validated by the polling agents and observers. The VVPAT slips are also cross verified with the EVM results.

Given that the results are declared primarily based on the EVM counts with only a limited verification of the VVPAT slips, the system is not end-to-end verifiable. A full manual tally from the VVPATs is prohibitively expensive due to the diversity and scale of the electorate. Moreover, it is difficult for the public to determine whether a full hand count was conducted accurately. Concerns have been raised about the security of the EVMs and the possibility of tampering. Researchers have found vulnerabilities in the EVMs [24] which allows the votes to be changed both at polling time and after polling. Citizen's Commissions of Elections (CCE), among its recommendations, suggest moving towards end-to-end verifiable systems [16].

4.2 Extending Indian EVMs with TallyGuard

Figure 1 shows the extension of the existing EVMs with TallyGuard. As mentioned previously, the public key used in the hashing process is loaded onto the EVMs during the pre-election setup. For ease of deployment and verification, the same public key may be loaded onto all the EVMs used in a particular constituency. The public key is released to the public before the polling day. We extend the EVM with functionality for hash computation. This receives the plaintext vote and generates the hash for the vote. Thus, for each vote, the EVM

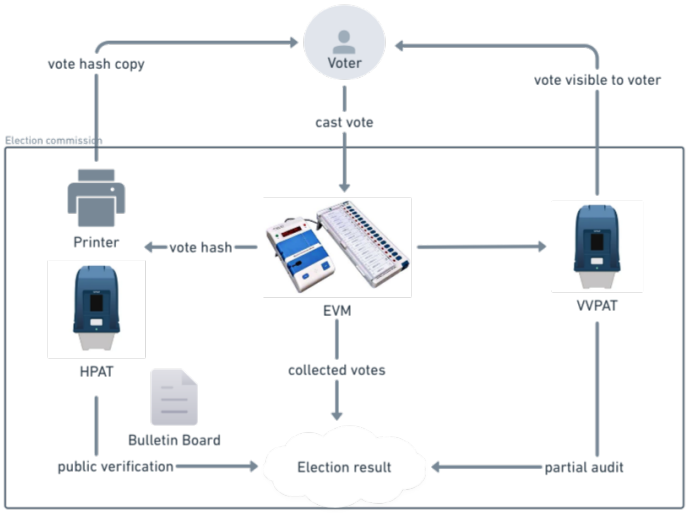

Fig. 1. Deployment architecture of TallyGuard in Indian general elections

now also stores the vote hash. In addition, the EVM also maintains the aggregate nonce vector.

We add a Hash Paper Audit Trail (HPAT) printer to the EVM. The HPAT printer prints vote hash on a paper slip along with a UUID that uniquely determines the voter. This UUID could be the same as the unique voter ID issued by the election commission. The HPAT slip is printed and shown to the voter for a few seconds and is securely stored in a box similar to VVPAT. A copy of the printed slip, with the vote hash and the UUID, is given as an acknowledgement to the voter. We also extend the pre-election audit of the EVMs to report the vote hashes and the aggregate random nonces. With the already revealed public key, the mock poll results are validated by the observers.

After the election, the EVMs, VVPATs and HPATs are transported to a secure location where they are kept until the counting day. During the counting, the election commission releases the results based on the counts from the EVMs. Alongside, the election commission also releases the vote hashes and the aggregate nonce vectors in a public bulletin board. Each voter can check that the vote hash is included in the public bulletin board. Any member of the public can verify the correctness of the results using the Theorems 1 and 2. HPAT may be used by the election commission to respond to a voter who may challenge that their vote hash was not recorded correctly.

We have prototyped TallyGuard using as a MirageOS Unikernel [17] written using the memory-safe OCaml programming language, running on top of the open-source security-enhanced Shakti RISC-V processor [11]. The use of MirageOS reduces the trusted computing base of the platform to a bare mini-

mum. OCaml programming language avoids many common memory errors by construction. The security-enhanced Shakti processor prevents many common programming errors becoming security holes. We believe that using open-source components both for software and hardware is essential for building trust in the implementation of TallyGuard. Overall, we believe that the proposed extension of the existing EVMs with TallyGuard is straight-forward and can be done with minimal changes to the existing infrastructure and processes.

5 Related work

TallyGuard follows the long line of end-to-end verifiable election systems work. Our specific aim was to design a system that is simple to extend to existing DRE-based voting systems. Given that our aim is to deploy the system to general elections, we would like to avoid any major changes to the process side of running the elections. Given that the election commission is currently a trusted stakeholder in conducting free and fair elections, we chose to not go for distributed trust and threshold encryption systems such as ElectionGuard [4], Belenios [7], StarVote [2] and Selene [21]. In these systems, the secret key is divided among multiple trustees and the decryption can only be done when all the trustees come together. The system assumes that at least one honest trustee exist to avoid decryption of individual votes. Such a system is more complex in terms of process than the current system. In the case of TallyGuard, the private key is destroyed and thus no decryption is possible. While the election commission still runs the elections, the public can verify the correctness of the results without having the ability to decrypt individual votes.

To gain cast-as-intended feature [5], we may use VVPAT or Benaloh challenge [3]. Benaloah challenge is a voter-initiated audit procedure, which provides choice for the voter to either challenge the encrypted vote for audit or cast the encrypted vote. Once audited that encrypted vote cannot be cast and a new encrypted vote needs to be prepared. Given the complexity of the Benaloah challenge, a casual voter may not be able to understand the process and may end up missing out on casting the vote.

Many other systems utilize homomorphic encryption for voting. Helios [1] is a Internet voting system where the votes are collected as encrypted votes from the users. For the result computation, the collected votes are homomorphically added and decrypted. In the online voting system, coercion resistance is a critical issue and Helios provides coercion explicitly to make it apparent. Damgård et al. [10] propose a generalization of Paillier encryption based electronic voting. The system uses encryption, homomorphic addition and decryption of the votes and extends the system using threshold encryption.

DRE-i [13] and DRE-ip [14,22] voting protocols are closest to TallyGuard in that they do not require trustworthy authorities to perform tallying operations. In the DRE-ip system, the vote receipt are generated using random numbers whose aggregate sum is used for verification. It uses Beneloah challenge to get cast-as-intended guarantee. Reversing the vote receipts is computationally

hard under the decisional Diffie-Hellman (DDH) assumption [12]. In TallyGuard, we start with DCRA assumption [19] holds for the secrecy of the vote hashes and then extend it to post-quantum secure hash functions. While TallyGuard does not use Benaloah challenge for cast-as-intended verification, we can extend TallyGuard to add this support similar to DRE-ip. Compared to DRE-ip, TallyGuard is extensible in the homomorphic encryption scheme. We have shown that TallyGuard can use either Pailier or Regev encryption. This flexibility allows a deployed TallyGuard system to switch to a different encryption scheme without changing the process model. We have also shown that TallyGuard is post-quantum secure, whereas no such claims are made for DRE-ip.

6 Limitations, Conclusions and Future Work

In this paper, we have presented TallyGuard, a verifiable electronic voting system that can be appended to existing offline DRE-based systems. Going back to the requirements listed in Section 1, TallyGuard satisfies the requirements as follows:

1. **Cast-as-Intended:** VVPAT ensures that the vote cast by the voter is the same as the vote intended by the voter. Mohanty et al. [18] observe that the current VVPAT auditing for Indian elections is not sufficient and propose an approach to conduct risk-limiting audits (RLA) of the ourcome of Indian elections. RLA is orthogonal and can be used in conjunction with TallyGuard.
2. **Recorded-as-Cast:** The voters can verify that their vote hash is included in the public bulletin board.
3. **Coercion Resistance and Voter Privacy:** The system has no ability to decrypt the individual votes from the vote hash.
4. **Tallied-as-Cast:** The result validation can be performed based on declared results, the public key (released before the election), the hashes from the public bulletin board and the aggregate random nonces (released along with the results).
5. **Deployability:** We have shown in Section 4 that TallyGuard can be deployed in a manner that is compatible with the current election process in India.

We have built a prototype of the system using MirageOS Unikernel running on top of security-enhanced Shakti RISC-V processor. The entire software and hardware stack is fully open-source and we plan to make the hardware and software stack available to the public. We believe that public scrutiny of the software and hardware stack is essential for trust in the system.

While the tallied-as-recorded guarantee is software independent – the results can be validated without relying on the software developed by the election commission – the other parts of our system still rely on trusting the hardware and

software, despite them being open-source. It is well-known that bugs in hardware and software may be exploited by attackers to make the system operate outside its specification, such as leaking the discarded private key at the point of key generation, leaking the random nonces used for encrypting individual votes, storing the association between the voter and their vote, etc. We plan to explore the use of formal verification for proving the correctness of the application software, using formally verified compiler such as CompCert [15] to ensure that the compilation of the software preserves the semantics of the source program, using hardware security techniques such as SecureBoot to ensure that the software running on the hardware is the software that was intended to run, etc.

As a next step, we plan to discuss the deployment of TallyGuard with the Election Commission of India and other stakeholders. We also plan to conduct a pilot deployment of the system in a small election to gather feedback from the voters and election officials. We believe that the deployment of TallyGuard will help in increasing the trust of the voters in the election process and will help in increasing the participation of the voters in the election process.

References

1. Adida, B.: Helios: Web-based Open-Audit voting. In: 17th USENIX Security Symposium (USENIX Security 08). USENIX Association, San Jose, CA (Jul 2008), https://www.usenix.org/conference/17th-usenix-security-symposium/helios-web-based-open-audit-voting
2. Bell, S., Benaloh, J., Byrne, M.D., Debeauvoir, D., Eakin, B., Kortum, P., McBurnett, N., Pereira, O., Stark, P.B., Wallach, D.S., Fisher, G., Montoya, J., Parker, M., Winn, M.: STAR-Vote: A secure, transparent, auditable, and reliable voting system. In: 2013 Electronic Voting Technology Workshop/Workshop on Trustworthy Elections (EVT/WOTE 13). USENIX Association, Washington, D.C. (Aug 2013), https://www.usenix.org/conference/evtwote13/workshop-program/presentation/bell
3. Benaloh, J.: Ballot casting assurance via voter-initiated poll station auditing. In: Proceedings of the 2007 Electronic Voting Technology Workshop (June 2007), https://www.microsoft.com/en-us/research/publication/ballot-casting-assurance-via-voter-initiated-poll-station-auditing/
4. Benaloh, J., Naehrig, M., Pereira, O., Wallach, D.S.: ElectionGuard: a cryptographic toolkit to enable verifiable elections. In: 33rd USENIX Security Symposium (USENIX Security 24). pp. 5485–5502. USENIX Association, Philadelphia, PA (Aug 2024), https://www.usenix.org/conference/usenixsecurity24/presentation/benaloh
5. Benaloh, J., Rivest, R., Ryan, P.Y.A., Stark, P., Teague, V., Vora, P.: End-to-end verifiability (2015), https://arxiv.org/abs/1504.03778
6. Bureau, P.I.: Largest electorate for General Elections - over 96.88 crore electors registered across the country — pib.gov.in, https://pib.gov.in/PressReleasePage.aspx?PRID=2005189
7. Cortier, V., Gaudry, P., Glondu, S.: Belenios: a simple private and verifiable electronic voting system. In: Guttman, J.D., Landwehr, C.E., Meseguer, J., Pavlovic, D. (eds.) Foundations of Security, Protocols, and Equational Reasoning - Essays Dedicated to Catherine A. Meadows, LNCS, vol. 11565, pp. 214–238. Springer

(2019). https://doi.org/10.1007/978-3-030-19052-1_14, https://inria.hal.science/hal-02066930
8. Crimmins, B.L., Narayanan, D.Y., Springall, D., Halderman, J.A.: Dvsorder: ballot randomization flaws threaten voter privacy. In: Proceedings of the 33rd USENIX Conference on Security Symposium. SEC '24, USENIX Association, USA (2024)
9. Culnane, C., Ryan, P.Y.A., Schneider, S., Teague, V.: vvote: A verifiable voting system. ACM Trans. Inf. Syst. Secur. **18**(1) (Jun 2015). https://doi.org/10.1145/2746338, https://doi.org/10.1145/2746338
10. Damgård, I., Jurik, M., Nielsen, J.B.: A generalization of paillier's public-key system with applications to electronic voting **9**(6), 371–385. https://doi.org/10.1007/s10207-010-0119-9, http://link.springer.com/10.1007/s10207-010-0119-9
11. Das, S., Unnithan, R.H., Menon, A., Rebeiro, C., Veezhinathan, K.: Shakti-ms: a risc-v processor for memory safety in c. In: Proceedings of the 20th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems. p. 19–32. LCTES 2019, Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3316482.3326356, https://doi.org/10.1145/3316482.3326356
12. Diffie, W., Hellman, M.: New directions in cryptography. IEEE Transactions on Information Theory **22**(6), 644–654 (1976). https://doi.org/10.1109/TIT.1976.1055638
13. Hao, F., Kreeger, M.N., Randell, B., Clarke, D., Shahandashti, S.F., Lee, P.H.J.: Every vote counts: Ensuring integrity in Large-Scale electronic voting. USENIX Journal of Election Technology and Systems (JETS) (3), 1–25 (Aug 2014), https://www.usenix.org/jets/issues/0203/hao
14. Hao, F., Wang, S., Bag, S., Procter, R., Shahandashti, S.F., Mehrnezhad, M., Toreini, E., Metere, R., Liu, L.Y.: End-to-End Verifiable E-Voting Trial for Polling Station Voting. IEEE Security & Privacy **18**(6), 6–13 (Nov 2020). https://doi.org/10.1109/MSEC.2020.3002728, https://ieeexplore.ieee.org/document/9152966/
15. Leroy, X.: Formal verification of a realistic compiler. Commun. ACM **52**(7), 107–115 (Jul 2009). https://doi.org/10.1145/1538788.1538814, https://doi.org/10.1145/1538788.1538814
16. M., L., W., H., A., K., P., P., J., D., S., B., M.G., D., Banerjee, S.: Citizens' commission on elections' report on evms and vvpat (2022)
17. Madhavapeddy, A., Mortier, R., Rotsos, C., Scott, D., Singh, B., Gazagnaire, T., Smith, S., Hand, S., Crowcroft, J.: Unikernels: library operating systems for the cloud. SIGARCH Comput. Archit. News **41**(1), 461–472 (Mar 2013). https://doi.org/10.1145/2490301.2451167, https://doi.org/10.1145/2490301.2451167
18. Mohanty, V., Culnane, C., Stark, P.B., Teague, V.: Auditing indian elections. In: Krimmer, R., Volkamer, M., Cortier, V., Beckert, B., Küsters, R., Serdült, U., Duenas-Cid, D. (eds.) Electronic Voting. pp. 150–165. Springer International Publishing, Cham (2019)
19. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: International Conference on the Theory and Application of Cryptographic Techniques (1999), https://api.semanticscholar.org/CorpusID:9483611
20. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography (2024), https://arxiv.org/abs/2401.03703

21. Ryan, P.Y.A., Roenne, P.B., Iovino, V.: Selene: Voting with transparent verifiability and coercion-mitigation. Cryptology ePrint Archive, Paper 2015/1105 (2015), https://eprint.iacr.org/2015/1105
22. Shahandashti, S.F., Hao, F.: DRE-ip: A Verifiable E-Voting Scheme Without Tallying Authorities. In: Askoxylakis, I., Ioannidis, S., Katsikas, S., Meadows, C. (eds.) Computer Security - ESORICS 2016, vol. 9879, pp. 223–240. Springer International Publishing, Cham (2016). https://doi.org/10.1007/978-3-319-45741-3_12, https://link.springer.com/10.1007/978-3-319-45741-3_12, series Title: Lecture Notes in Computer Science
23. Wikipedia: Electronic voting in India - Wikipedia — en.wikipedia.org, https://en.wikipedia.org/wiki/Electronic_voting_in_India#Design_and_technology
24. Wolchok, S., Wustrow, E., Halderman, J.A., Prasad, H.K., Kankipati, A., Sakhamuri, S.K., Yagati, V., Gonggrijp, R.: Security analysis of india's electronic voting machines. In: Proceedings of the 17th ACM Conference on Computer and Communications Security. p. 1–14. CCS '10, Association for Computing Machinery, New York, NY, USA (2010). https://doi.org/10.1145/1866307.1866309, https://doi.org/10.1145/1866307.1866309
25. Zhang, A., Li, Z.: A new lwe-based homomorphic encryption algorithm over integer. In: 2021 International Conference on Computer Information Science and Artificial Intelligence (CISAI). pp. 521–525 (2021). https://doi.org/10.1109/CISAI54367.2021.00106