VSPReP: Verifiable, Secure and Privacy-Preserving Remote Polling with Untrusted Computing Devices

Amna Qureshi(✉), David Megías, and Helena Rifà-Pous

Internet Interdisciplinary Institute (IN3), Universitat Oberta de Catalunya (UOC), Barcelona, Spain
{aqureshi,dmegias,hrifa}@uoc.edu

Abstract. Internet-based polling systems allow voters to cast their votes at any time during the polling period, from any Internet-connected computing device anywhere in the world. Security is an important feature of such systems that should address inherent concerns, such as secrecy of vote, anonymity and unlinkability of voter, voter coercion, secrecy of intermediate results, verifiability, auditability, and poll integrity. Another major concern is that an infected voting device with a malicious program (e.g., virus, malware) could take control over the vote casting process and make unauthorized and potentially undetected modifications to the voter's voting choices, and, hence, should not be trusted. In this paper we present VSPReP, a verifiable, secure and privacy-preserving remote polling (e-poll) system, which provides vote's privacy and poll integrity, prevents double voting, enables multiple voting (within the allowed polling period), and achieves verifiability (cast-as-intended and tallied-as-recorded) and uncoercibility in the presence of an untrusted voting device. This paper presents a general design of VSPReP and describes its workflow during three polling phases: pre-polling, polling and post-polling. It also analyzes the security properties of VSPReP and evaluates its performance in terms of computational and cryptographic costs. The experimental results show that the average time a voter takes to cast his/her vote is less than 45 secs, thus demonstrating the practicality of VSPReP.

Keywords: Remote polling · Malware detection · Privacy Verifiability

1 Introduction

In traditional elections, a voter presence is necessary to take part in an election/poll and cast a vote. With the rise and popularity of the Internet and mobile phones, elections/polls could be conducted remotely. Today, a trend towards electronic and Internet voting can be observed, e.g., online polls and surveys are popular in social networks, forums and newspapers. Similarly, till date, 14 countries have conducted election trials to enable voters to cast votes on the Internet using their own computing devices [14].

Internet-based voting system is based on the voter's computing device (smartphone, tablet, desktop PC, etc.), the Internet, and the voting system. The voter's computing device casts the votes that are sent across the Internet to the voting system, where they are stored and tallied. These three different environments and the information shared between them are vulnerable to various attacks [18], such as voter coercion (a voter is put under pressure or is threatened by a coercer to vote in a particular manner) or vote buying (a voter is offered monetary benefits by a vote buyer to vote in a particular way, or not at all), vote modification due to an infected voting device (a malicious program such as a malware or virus may cause unauthorized and potentially undetected alterations to voter's selected voting choices), theft/forgery of voter identity (an attacker with an access to authentication credentials could cast votes using the identities of a legitimate voter), double voting (an eligible voter may cast multiple votes using his/her authenticated credentials), a coalition of malicious participants (involved parties may collude to alter or eliminate any voter's vote, or cast fake ballots on the behalf of authenticated voter), and disclosure of partial vote tally before the end of the voting period.

Designing a secure Internet-based voting system has become a considerable topic of discussion in the scientific community. A number of schemes have been implemented and deployed in real-world, e.g., Prêt à Voter [17], and Helios [1], which ensure vote privacy as well as verifiability in the presence of untrusted authorities. However, these systems assume that the voting device is trusted for privacy and verifiability. This assumption is unrealistic because a voting device might be controlled by an attacker or host a malicious program. To resolve this problem, many e-voting protocols are proposed to provide three types of verifiability: (1) cast-as-intended verifiability [3,11] that provides a voter with means to make sure that the vote cast by his/her voting device contains the intended voting option, and that no changes have been performed, (2) tallied-as-cast verifiability [12] that allow voters, auditors and third party observers to check that votes tallied corresponds to the cast votes, and (3) end-to-end verifiability [5] that provides both cast-as-intended and tallied-as-cast verifiability. Here, the tallied-as-cast verifiability is divided into two phases: recorded-as-cast (voters can check that their cast votes have been properly recorded in the ballot box) and tallied-as-recorded (voters, auditors and third party observers can verify that the votes published on the BB are correctly included in the tally, without knowing how any voter voted). In the literature, there exists a few protocols that use return codes to provide cast-as-intended verifiability [2,3,9,11], and a Bulletin Board to provide tallied-as-cast verifiability [12]. These return code-based systems send a code sheet containing pre-generated return codes, and finalization codes to the registered voter over a secondary channel (postal mail) before the voting phase. During voting, when the voter selects his/her voting choices and the voting device submits an encrypted vote to the remote voting server, the voting authorities calculate or retrieve return codes corresponding to voter's choices. These codes are sent back to the voter who compares them with the pre-generated return codes printed on his/her code sheet against the selected choices. If matches, the voter finalizes the vote casting process by using finalization codes. The issue of these schemes is that they assume the voting device is not compromised and supports single vote casting.

The scientific community has published a lot of research work in designing secure voting systems for national-level or big elections, and less attention has been paid to develop secure e-polling systems (low-risk or small-level public-opinion systems, where reasonable level of security, privacy, and functionality should be provided to the voter).

In this paper, we propose an e-polling system, VSPReP, inspired by return codes-based protocols [3,6,9] to provide cast-as-intended verifiability in the presence of untrusted voting devices. Also, VSPReP provides recorded-as-cast verifiability (while preserving the privacy of the voter), poll integrity, non-coercibility, resistance against collusion of voting authorities, and supports multiple voting within an allowed polling period, while preventing double voting. The security analysis of the e-polling protocol, and the experimental results of the polling phase (implemented on Java programming language) are presented to show that VSPReP provides a balance between security and functionality.

The rest of the paper is organized as follows. In Sect. 2, we provide the building blocks of VSPReP. Section 3 describes VSPReP in detail. The security analysis and experimental results are discussed in Sect. 4. Finally, Sect. 5, concludes the paper.

2 Building Blocks

A. Distributed ElGamal Cryptosystem: In a distributed cryptosystem, a set of agents cooperate to perform decryption on encrypted messages so as to provide confidentiality by preventing any single agent from decrypting messages. In VSPReP, distributed ElGamal cryptosystem proposed in [10] is used to provide voters' privacy. Distributed ElGamal cryptosystem is a set of three protocols: key generation (*KeyGen*), encryption (*Enc*), and decryption(*Dec*). In *KeyGen* algorithm, a subgroup $\mathcal{G}_p$ is taken on as input which has a generator g of order q of elements in $\mathbb{Z}_p^*$ (a message space of the cryptosystem), where p and q are two large numbers with $p = 2kq + 1$ for some integer constant $k > 0$. *KeyGen* outputs ElGamal public key $y = g^x$ (global and known to all parties), and a secret key x that is shared among t polling organizers $(PO_1, \ldots, PO_t)$ using a polynomial f of degree l over $\mathbb{Z}_q$ such that each polling organizer holds a share $x_i = f(i)$. In *Enc* algorithm, a message $m \in \mathcal{G}_p$, y, and a randomly chosen $r \in \mathbb{Z}_q$ are taken as inputs to compute a cipher-text c: $c = (c_1, c_2) = (g^r, y^r.m)$. For decryption of c, *Dec* algorithm requires all polling organizers to compute decryption shares $d_i = {c_1}^{x_i}$ to output a plain-text message m. To provide verifiability, non-interactive zero-knowledge proofs are computed during *KeyGen* and *Dec* protocols.

B. Pseudo-random Function Based on Decisional Diffie-Hellman (DDH): A pseudo-random function (PRF) is a deterministic-keyed function $F : \mathcal{K}$ x $\mathcal{X} \rightarrow \mathcal{Y}$ (where $\mathcal{K}$ is the set of keys, $\mathcal{X}$ is the domain, and $\mathcal{Y}$ is the range) guaranteeing that a computationally bounded adversary having access to PRF's outputs at chosen points, cannot distinguish between the PRF and a truly random function mapping between the same domain and range as the PRF. In VSPReP, we use a variant of PRF, a key homomorphic PRF (F_{DDH} based on DDH), proposed by Naor et al. [15]. A PRF is key homomorphic if given $F(k_1, m)$ and $F(k_2, m)$, there is a procedure that outputs $F(k_1 \oplus k_2, m)$, where $\oplus$ denotes group operation on k_1 and k_2. F_{DDH} is constructed by considering a cyclic group $\mathcal{G}_p$ of order q, and a hash function $\mathcal{H}_1$: $\mathcal{X} \rightarrow \mathcal{G}_p$ modeled as a random oracle. F_{DDH} is defined as: $F_{\mathrm{DDH}}(k, m) \leftarrow \mathcal{H}_1(m)^k$ with the following homomorphic property, $F_{\mathrm{DDH}}(k_1 + k_2, m) = F_{\mathrm{DDH}}(k_1, m) \cdot F_{\mathrm{DDH}}(k_2, m)$. F_{DDH} is a secure PRF in the random oracle model assuming the DDH assumption holds in $\mathcal{G}_p$.

C. Verifiable Mixnet: Verifiable mixnet is used to provide an anonymous and verifiable tally in electronic voting systems. Verifiable mixnet enables a collection of trustworthy servers to take as input an ordered set of cipher-texts $E = E_1, E_2, \ldots, E_N$ to be re-encrypted using a new randomization value without changing the decryption process. The output is an ordered set of encryptions $E' = E'_{\pi(1)}, E'_{\pi(2)}, \ldots, E'_{\pi(N)}$ (where $E'_{\pi(N)}$ is a re-encryption of E_N, and π is a uniformly random and secret permutation), and non-interactive zero-knowledge proofs π_{mix_t} (where $t = 1, \ldots, N$) of correct mixing. Thus, this re-randomized encryption prevents an adversary to determine the link between the output and the input cipher-texts. The link between elements from input and output is only retrieved in case of conspiring mix-nodes. Verifiability is provided by π_{mix_t}, which is checkable by any party and demonstrates that E' is correctly constructed. The tallying phase (Sect. 3.4) of VSPReP employs the verifiable mixnet proposed in [20].

D. Digital Signature Scheme: A digital signature scheme (e.g., RSA, DSA) is used to provide data integrity, data origin authentication and non-repudiation. In our proposed system, we have used the RSA signature [4] that is made up of three algorithms, (*Gen*, *Sign*, *Verify*), for generating keys, signing, and verifying signatures, respectively. *Gen* is a key generation algorithm that creates an RSA pubic key pk ($pk = (n, e)$), and a corresponding RSA private key sk ($sk = d$), where n is a product of two large distinct prime numbers p and q, e is a public exponent (a randomly generated integer with $1 < e < \phi$, where $\phi = (p-1)(q-1)$), and d is a private unique integer with $1 < d < \phi$. *Sign* is a probabilistic signature algorithm that takes a message m as an input, produces a hash H of m, and then computes a signature S on hash value (H_s) using sk. *Verify* is a deterministic verification algorithm that takes pk and a signature S as inputs to extract hash H_s from S. Also, it computes hash on the received message to generate another hash value (H_v), and compares it with H_s for verification purposes. If both hashes are identical, S is considered valid, otherwise invalid.

E. Crypto MAC: Message Authentication Code (MAC) is a cryptographic primitive that relies on a pseudorandom function to provide authentication, and verification of received messages. A specific type of MAC, the keyed-hash message authentication code (HMAC), is used to provide data integrity and authenticity of the message. HMAC is obtained by using a cryptographic hash function (e.g., SHA256) over the data (to be authenticated) in combination with a secret (symmetric) key. The cryptographic strength of a HMAC depends on the properties of the underlying hash function. The ballot processing phase (Sect. 3.4) and polling codes generation phase (Sect. 3.4) of VSPReP relies on the HMAC algorithm described in [13].

F. Non-Interactive Zero Knowledge Proofs: A non-interactive zero knowledge proof (NIZKP) is a variant of zero knowledge proof that does not require an interaction between the prover and the verifier. The prover computes and sends a statement to the verifier, who either accepts or rejects it. NIZKPs can be obtained in the random oracle using Fiat-Shamir heuristic [8]. To provide verifiability in VSPReP, we have used the following proofs in different phases of the polling: (1) proof of correct encryption based on Schnorr protocol [19] (polling phase), (2) proof-of-equality of discrete logarithms based on Chaum-Pederson protocol [7] (polling phase), (3) proof of correct decryption of ElGamal ciphertexts (π_{dec_t}) (mix and tallying phase), and (4) proof of correct mixing (π_{mix_t}) of ElGamal encryptions in the mixnet (mix and tallying phase).

3 VSPReP Model

This section describes the design and functionality of VSPReP. In Sect. 3.1, we describe the role of each entity. Section 3.2 defines the functionality requirements and the security assumptions. An attack model is described for VSPReP in Sect. 3.3. Section 3.4 describes three phases of VSPReP in detail.

3.1 VSPReP Entities

VSPReP consists of eight basic entities. The functionality of each entity is defined as follows: **(1)** The **voter**(V_k) is a participant who has a valid credential obtained from the credential issuer to cast a vote ($k = 1, \dots, N$, where N is equal to maximum voters allowed in polling). **(2)** The **voting device** (VD$_S$) is a computing device responsible of casting a ballot given the options selected by V_k. A voter V_k can use as many as S computing devices to cast his/her vote. Besides VD$_S$, V_k uses another computing device as a validation device to receive return and confirmation codes. **(3)** The **polling organization** is a trusted entity that is in-charge of setting up the poll (poll questions and their corresponding voting options, etc.), tallying the votes and publishing the results of the poll. The polling organization consists of t polling organizers: PO$_1 \dots$ PO$_t$. It is assumed that out of t POs, there is one main PO who manages the remaining POs. **(4)** The **credential issuer (CI)** is a trusted third party that is responsible for authentication and registration of the voter. It provides authenticated voters with the necessary polling credentials (keys and pseudo-identities). **(5)** The **bulletin board (BB)** is a publicly verifiable entity where the results of various steps of the polling process including the final polling result are published by the authorized entities. All the entities of VSPReP have read-only access to BB, whereas some parties have write-only and append-only access to BB. No party is allowed to delete the existing data. **(6)** The **polling server (PS)** checks the correctness of the ballots cast by the authenticated voters, updates, records and stores these ballots into the ballot box. **(7)** The **code generator (CG)** is an entity that manages multiple polling code generators (PCG). In VSPReP, we have assumed six $PCG_{\mathbb{X}}$ ($\mathbb{X} = 1, \ldots, 6$) that are responsible of generating return codes, acknowledgment and confirmation codes to be used in the polling phase. Also, each PCG generates a mapping table to map long-length return codes to small-length return codes. **(8)** The **printing facility (PF)** is in-charge of printing voting options along with their corresponding return codes, polling code sheet identity (PCS_{ID}), acknowledgment and confirmation codes. Also, PF in cooperation with CI, delivers polling card sheets to the authenticated voters only.

3.2 Design Requirements and Security Assumptions

In this section, the design requirements and assumptions of VSPReP are described.

A. Design Requirements: In the following, the design requirements related to the construction of VSPReP are defined: **(1)** Only an authenticated voter can use VSPReP on his/her lightweight computing device to cast his/her votes for a maximum of three times. Only the last vote cast by the voter (within an allowed voting period) is considered valid. Double voting by the same voter is not allowed. **(2)** A voter should use the same pseudo-identity (issued by CI at the time of registration) in three rounds of the poll. In case of a new pseudo-identity request, all the previous votes of the voter shall be revoked. **(3)** Three votes from the same voter within a permitted polling period shall be linked together by a tag, which is a poll-specific pseudonym signed by the PS. **(4)** No vote can be linked to the identity of the voter who has cast it. **(5)** All ballots must remain secret while polling is in progress. A voter can read the contents of the BB once the polling phase is finished. **(6)** Since the voting device of the voter is untrusted, the integrity of the vote must be guaranteed, i.e. a verification mechanism is required that should prevent vote's manipulation by the malware-affected device. **(7)** A voter should not be able to provide a proof of his/her vote to any other entity. **(8)** A voter cannot be coerced by a coercer to cast a vote for a specific voting option or abstain from voting. **(9)** No entity can gain any knowledge about the tally before the start of the vote counting phase. **(10)** After the polling phase and before the start of the tallying phase, a voter should be able to verify that the vote cast by his/her voting device corresponds to what he/she intended to cast in the voting phase. **(11)** After the tallying phase, the results should be published on the BB and can be verified by the voter, auditors or passive observers that the final tally is correctly computed from the votes that were cast. Also, a voter should be able to verify that his/her vote was correctly included in the tallying phase. **(12)** The polling system should be efficient and scalable.

B. Design and Security Assumptions: The underlying design and security assumptions of our scheme are described as follows: **(1)** The poll consists of multiple choice questions in which each voter should mark his/her preferences (selecting one option per question) and order them sequentially. **(2)** A voter is allowed to cast his/her vote three times within the allowed voting period using his/her voting device. A tag is used to identify different votes sent by a single voter within the voting period. **(3)** VSPReP assumes a TLS channel between a voting device and the polling server during polling phase. **(4)** The polling protocol of VSPReP depends on the voter using two computing devices (one for casting vote, i.e. a voting device, and another for receiving return and confirmation codes, i.e. a validation device). **(5)** The polling card sheets are provided to each voter through a secure communication channel (post, email, etc.) by a printing facility. Once the polling card sheets are delivered to the voters, PF destroys all the information related to these sheets. **(6)** The existence of PKI is assumed such that any entity who uses the public key of another participant knows that this key belongs to a legitimate party. The RSA and ElGamal key generation is performed offline to generate key pairs. **(7)** Cryptographic primitives and constructions used in VSPReP are secure and verifiable. **(8)** The return, acknowledgment and confirmation codes are composed of 6, 6 and 8 alphanumeric digits, respectively. We have assumed the use of all upper-case and lower-case letters, and digits (0–9). **(9)** Cast-as-intended verifiability depends on the assumption that the voting device and the polling server cannot be malicious simultaneously. **(10)** Generation of polling card sheets is an offline process. In our experimental case, we assume each polling code generator has 50 pre-generated keys (1024-bits) to assist more than $10,000$ users. **(11)** Each voter has access to the general parameters and the public keys, which are made available by the polling server, the polling organizers, the printing facility, the code generator and the polling code generators. **(12)** Six polling code generators are assumed in generation of the polling card sheets. The reason of considering multiple PCGs in VSPReP is to make the system scalable. With an increase in the number of PCGs in the system, the computational cost of generating temporary keys in the return codes generation phase is reduced.

3.3 Threat Model

This sub-section highlights an attack model for VSPReP related to coercion resistance, double voting, vote manipulation by a malicious voting device, and the voter's privacy. The security of the system against these attacks is discussed in Sect. 4.1.

A. Voter Coercion: In voter coercion, the voter may be threatened by a coercer to vote his/her choice of voting options. Once a vote casting phase finishes, the election authority may want to provide a receipt to the voter to allow individual verifiability. The vote coercion attack is possible as long as the voters are able to prove to the coercer how they voted.

B. Double Voting: Remote polling is vulnerable to electoral fraud due to the possibility of double voting, i.e. an authenticated but a malicious voter may request different polling credentials from the credential issuer to cast multiple ballots in the same poll.

C. Vote Modification by a Malicious Voting Device: In a remote electronic polling system, the voters input their vote choices on a privately owned computing devices. If the voting device is infected with a malware, it can modify the voter's choices covertly before these are submitted to the election authorities and, therefore, falsely recorded and counted by the election authority undetectably (without the voter's knowledge).

D. Coalition of Malicious Entities: The following three attacks describe the coalition of malicious VSPReP's participants: (a) A malicious VD$_S$ may form a coalition with PS to generate partial return codes (not corresponding to voter's encrypted voting options) undetectably; (b) PS and CG may collude to infer the voting choices selected by the voter; and (c) After sending the valid confirmation code to the voter, PS may collude with the CG to replace the voter's ballot in the ballot box with the colluded vote.

3.4 Overview of VSPReP

VSPReP, as shown in Fig. 1, consists of 3 phases: pre-polling, polling, and post-polling.

In the pre-polling phase, cryptographic keys and polling parameters are cooperatively generated by the POs of VSPReP. Also, a web address containing the list of voting options for each polling question, a unique poll identity, and a polling period, are generated by the POs. This url is only sent to the authenticated voters on request of CI. A voter gets registered to the system through the voter registration phase, in which the voter receives a unique and valid credential from CI after a successful authentication. Only the authenticated voters receive the polling card sheets (PCSs) (via mail) from the PF on CI's request. A PCS contains return codes, acknowledgment and confirmation codes, and is generated by the CG, six PCGs, and the PF.

During the polling phase, the authenticated voter uses his/her credential to input his/her voting options into VD_S that encrypts the selected voting options, computes and encrypts partial return codes, generates NIZKPs and forms a ballot. The ballot contains an encrypted ID, cipher-text of votes, NIZKPs, time stamp, session ID (a poll-specific pseudonym), and encrypted partial codes. VD_S sends the ballot to the PS via an anonymous and secure channel (TLS). Before

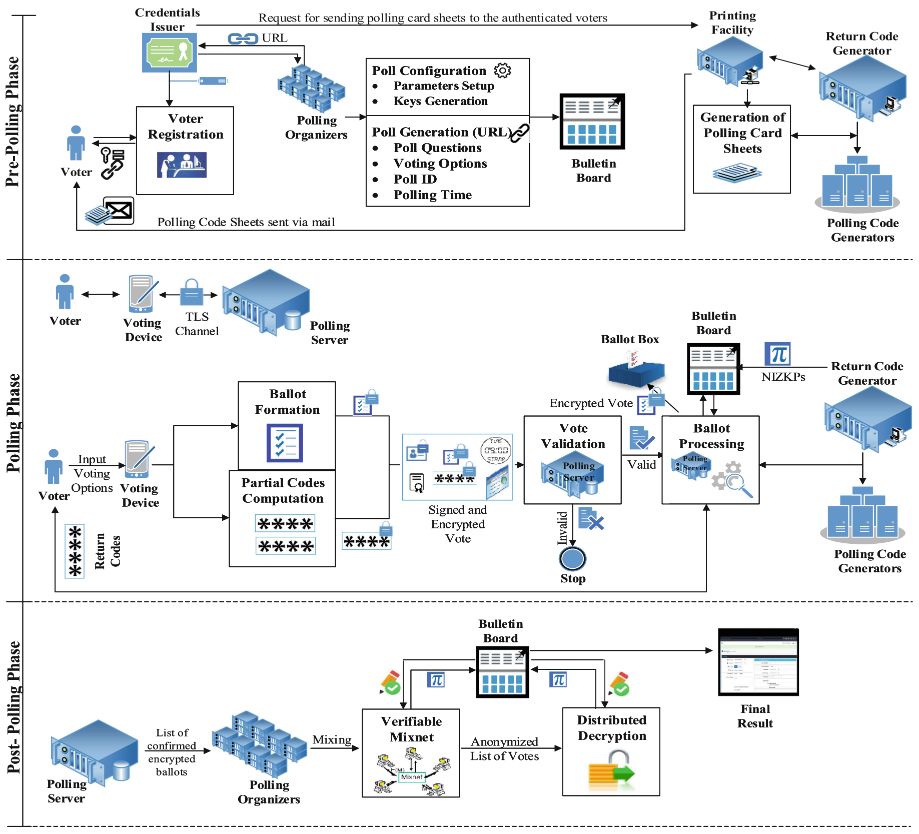

Fig. 1. Overview of VSPReP.

the post-polling phase, vote validation and ballot processing phases are performed to remove duplicate votes, and provide individual verifiability to the voter.

In the post-polling phase, POs input the list of confirmed encrypted votes into a verifiable mixnet that outputs an anonymized list of cipher-texts and NIZKPs (proofs of correct mixing). These cipher-texts are then distributedly decrypted by the POs to reveal the original votes, which are then published on the BB.

In this paper, we have described two phases (generation of a PCS, and a polling phase) in detail due to the fact that these two processes of e-poll protocol address our objectives of providing protection against malware (during polling), prevention of double voting, individual verifiability, and coercion resistance. The post-polling is similar to other voting schemes in the literature that employ mixnets to preserve anonymity of votes.

A. Pre-polling Phase: In this preliminary phase, V_k gets registered, and the polling system is configured: an e-poll is created, cryptographic parameters and keys are generated and published on the BB by POs, and PCS are generated and distributed to the authenticated voters.

I. Voter Registration: To be able to cast a vote, V_k must first register to VSPReP to obtain his/her polling credential from CI. V_k can prove his/her identity (e.g., eID card, a digital certificate issued by a trusted authority, verified email address) to CI, and obtains his/her polling credentials, i.e. a key pair (K_{pV_k}, K_{sV_k}) and a pseudo-identity (we abstract here from the details of authentication and assume that a secure authentication mechanism is used). The pseudo-identity is obtained through a successful run of an interactive protocol [16] between CI and V_k. This protocol results in a shared secret random value r_{V_k} between CI and V_k, which is used along with other identity details of V_k to generate a unique pseudo-identity.

II. Poll Configuration: During polling configuration phase, the polling cryptographic parameters (p, q, g) to be used in ElGamal cryptosystem and homomorphic PRF are defined and published. A cyclic $\mathcal{G}_p \subseteq \mathbb{Z}_p^*$ of quadratic residues modulo a safe prime $p = 2q + 1$ is chosen as a common group for all the cryptographic operations used in VSPReP. The key pairs of PF (K_{pPF}, K_{sPF}), PS (K_{pPS}, K_{sPS}), CG (K_{pCG}, K_{sCG}), and PCGs are generated. Also, PCGs create their joint public encryption key K_{pPCG} and a shared secret decryption key K_{sPCG} using distributed cryptosystem. Similarly, POs create a joint public encryption key and a shared secret decryption key for ElGamal encryption and decryption. Each PO creates its share of the key and posts the public part along with the proofs at BB. BB checks the proofs and combines the shares to form a public election key (K_{pPO}). A message encrypted under K_{pPO} can only be decrypted by K_{sPO} if all POs collaborate. POs and PS are provided with "write" and "append" access to the BB. CG is provided with "write-only" access to the BB. The voters are provided with "read-only" access to the BB. A poll description is generated by POs that contains a unique poll identifier, poll questions, voting options $v_j = \{A, B, C, D\}$ (small bit-length prime numbers $\in \mathcal{G}_p$) for each question, polling time period (t_p), and K_{pPO} to be used by the voters to encrypt their votes before casting them. This data is signed by the main PO, and is appended to the poll description. The link containing the poll description is sent to authenticated voters by CI after a successful registration.

III. Generation of Polling Card Sheets: For the generation of PCSs, CG, $PCG_{\mathbb{X}}$, and PF perform cryptographic operations using their respective key pairs. For proof-of-concept, it is assumed that there are 3 polling questions (Q_i with $i = 1, 2, 3$) with each Q_i having 4 voting options (v_j with $j = 1, 2, 3, 4$) represented by small bit-length prime numbers. For example, the following voting options for 3 questions are generated by the main PO, and are communicated to $PCG_{\mathbb{X}}$ before generation of PCSs: $v_{1j} = \{11, 13, 17, 19\}$, $v_{2j} = \{7, 29, 31, 41\}$ and $v_{3j} = \{433, 53, 5, 47\}$. Each $PCG_{\mathbb{X}}$ randomly picks up a key from a pool of 50 ($l = 1, \ldots, 50$) pre-generated keys. Also, each $PCG_{\mathbb{X}}$ generates a unique key $K_{sess_{\mathbb{X}}}$ of 256-bits. Figure 2 illustrates the following steps performed between CG, $PCG_{\mathbb{X}}$ and PF to generate PCSs.

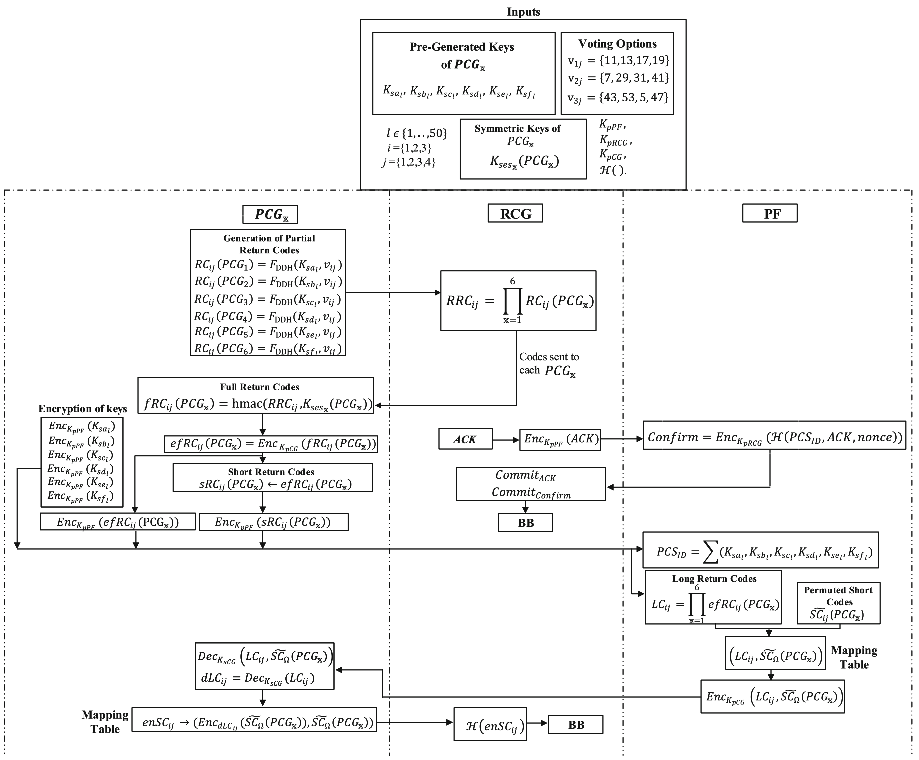

Fig. 2. Generation of polling card sheets.

(1) Each $PCG_{\mathbb{X}}$ calculates partial return codes $(RC_{ij}(PCG_{\mathbb{X}}))$ using PRF based on DDH assumption for each voting option v_j of Q_i. Each $PCG_{\mathbb{X}}$ uses its selected secret key to obtain $RC_{ij}(PCG_{\mathbb{X}})$ in the following way:

$$\begin{aligned} RC_{ij}(PCG_1) &= F_{\mathrm{DDH}}(K_{sa_l}), & RC_{ij}(PCG_2) &= F_{\mathrm{DDH}}(K_{sb_l}), \\ RC_{ij}(PCG_3) &= F_{\mathrm{DDH}}(K_{sc_l}), & RC_{ij}(PCG_4) &= F_{\mathrm{DDH}}(K_{sd_l}), \\ RC_{ij}(PCG_5) &= F_{\mathrm{DDH}}(K_{se_l}), & RC_{ij}(PCG_6) &= F_{\mathrm{DDH}}(K_{sf_l}), \end{aligned}$$

where $F_{\mathrm{DDH}}(K, v_{ij}) = \mathcal{H}_1(v_{ij})^K$. **(2)** For each code, $PCG_{\mathbb{X}}$ sends $RC_{ij}(PCG_{\mathbb{X}})$ to CG that computes product of the received partial codes, and sends the result (RRC_{ij}) to each $PCG_{\mathbb{X}}$, e.g., RRC_{11} that corresponds to voting option "1" of Q_1 is computed as: $RRC_{11} = \prod_{\mathbb{X}=1}^{6} RC_{11}(PCG_{\mathbb{X}})$. **(3)** Each $PCG_{\mathbb{X}}$ computes full return code using the received code RRC_{ij} (from CG), and the symmetric key in the following way: $fRC_{ij}(PCG_{\mathbb{X}}) = hmac(RRC_{ij}, K_{ses_{\mathbb{X}}})$. Then, $fRC_{ij}(PCG_{\mathbb{X}})$ is encrypted with K_{pPCG} to obtain encrypted return codes $(efRC_{ij}(PCG_{\mathbb{X}}))$ to be sent to PF. Also, each $PCG_{\mathbb{X}}$ generates small-length (64-bits codes) random codes $sRC_{ij}(PCG_{\mathbb{X}})$ that correspond to long (1024-bits $efRC_{ij}$) return codes $(sRC_{ij}(PCG_{\mathbb{X}}) \leftarrow efRC_{ij}(PCG_{\mathbb{X}}))$, and encrypts both $sRC_{ij}(PCG_{\mathbb{X}})$

and $efRC_{ij}(PCG_{\mathbb{X}})$ with the public key (K_{pPF}) of PF to be sent to PF. Also, each $PCG_{\mathbb{X}}$ encrypts its secret key used in computation of partial return codes $(RC_{ij}(PCG_{\mathbb{X}}))$ with K_{pPF}, and sends the encrypted key to PF. **(4)** CG generates a random Acknowledgment (ACK) code (64-bits code encoded with Extended ASCII encoding to 6 digits), which a voter uses in the polling phase to provide confirmation of the received return codes. CG encrypts ACK with K_{pPF}, and sends the encrypted code $(Enc_{K_{pPF}}(ACK))$ to PF. **(5)** When PF receives the sets of the return codes (both long and short), the encrypted keys, and $Enc_{K_{pPF}}(ACK)$, it computes the following:

(a) PF decrypts the received encrypted keys using its K_{sPF}, and generates a Polling Card Sheet ID: $PCS_{ID} = \sum(K_{sa_l}, K_{sb_l}, K_{sc_l}, K_{sd_l}, K_{se_l}, K_{sf_l})$.
(b) PF decrypts the received encrypted long return codes with K_{sPF} to compute a long code (LC_{ij}) for each voting option: $LC_{ij} = \prod_{\mathbb{X}=1}^{6} efRC_{ij}(PCG_{\mathbb{X}})$.
(c) PF decrypts $Enc_{K_{pPF}}(ACK)$ to obtain ACK. Also, PF decrypts encrypted short return codes to obtain plain-text short return codes $SC_{ij}(PCG_{\mathbb{X}})$.
(d) PF permutes $SC_{ij}(PCG_{\mathbb{X}})$ with a random permutation key ρ, and then randomly selects permuted $(\widetilde{SC_{ij}}(PCG_{\mathbb{X}}))$ codes, and pairs them with the encrypted long return codes such that it obtains $i \times j$ pairs of return codes to create a mapping table, e.g., in our proof-of-concept, $3 \times 4 = 12$ pairs of codes are generated: $(LC_{ij}, \widetilde{SC_{ij}}(PCG_{\mathbb{X}}))$. Each entry of the mapping table is then encrypted with K_{pPCG}, and sent to each $PCG_{\mathbb{X}}$.
(e) PF generates a confirmation number $(Confirm)$ by using PCS_{ID}, ACK, and a *nonce*: $Confirm = H(PCS_{ID}, ACK, nonce)$. $Confirm$ is used as a proof that the vote has been confirmed by the voter. PF encrypts $Confirm$ with K_{pCG}, and sends the encrypted code to CG.
(f) PF prints the randomly selected 12 short return codes $(\widetilde{SC_{ij}}(PCG_{\mathbb{X}})$ along with the corresponding voting options $\{A, B, C, D\}$, PCS_{ID}, $Confirm$, and ACK as a Polling Card Sheet.

(6) Upon receiving $Enc_{K_{pPCG}}(LC_{ij}, \widetilde{SC_{ij}}(PCG_{\mathbb{X}}))$ from PF, each $PCG_{\mathbb{X}}$ distributedly decrypts the encrypted entries (one time to decrypt the pair, and a second time to decrypt LC_{ij} to obtain long return codes (dLC_{ij})). The short return codes $(\widetilde{SC_{ij}}(PCG_{\mathbb{X}}))$ are encrypted with the corresponding long return codes dLC_{ij} to obtain $Enc_{dLC_{ij}}(\widetilde{SC_{ij}}(PCG_{\mathbb{X}}))$, which is paired with plain-text $\widetilde{SC_{ij}}(PCG_{\mathbb{X}})$ to create a mapping table that contains pairs $(i \times j)$ of return codes:

$$enSC_{ij} = (Enc_{dLC_{ij}}(\widetilde{SC_{ij}}(PCG_{\mathbb{X}})), \widetilde{SC_{ij}}(PCG_{\mathbb{X}})).$$

This mapping table is shared between CG and $PCG_{\mathbb{X}}$. **(7)** CG computes the hash of each pair in the mapping table, and publishes it on BB as commitments to the return codes. Also, CG computes commitments to ACK and $Confirm$ codes, and publishes $Commit_{ACK_i}$ and $Commit_{Confirm_i}$ on the BB (since 3 PCSs will be sent to each voter, thus, BB would contain 3 tables of commitments to the return codes, and $i = 3$ values of $Commit_{ACK_i}$ and $Commit_{Confirm_i}$ for each voter).

B. Polling Phase: Once the pre-polling phase is finished, each V_k may cast his/her ballot using his/her VD_S. VD_S of each V_k creates a ballot with the selected voting options of each polling question (Q_i), and submits it to PS. V_k can cast his/her ballot at most three times ($1 \leq t \leq 3$). Each V_k casts his/her ballot as follows: **(1)** VD_S sets up a TLS connection with PS. PS authenticates the VD_S and receives a session ID (a poll specific pseudonym PI_{Poll}) and a time-stamp (t_s) with the current time. **(2)** V_k selects one option for each Q_i, i.e. it inputs v_j of each Q_i into his/her VD_S. **(3)** VD_S computes a partial ballot as a product of V_k's selected options (v_{ij}): $B_{V_k} = \prod_{i=1}^{3} v_{ij}$. VD_S encrypts B_{V_k} with the joint public key of POs (K_{pPO}) to obtain ElGamal cipher-text: $(c_1, h_1) = Enc_{K_{pPO}}(B_{V_k})$. Also, VD_S generates NIZKP (π_{enc}) to prove knowledge of the randomness used for computing the encryption of B_{V_k}. **(4)** Additionally, V_k inputs a 3-digit (alphanumeric) random number for each voting option ($\gamma_1, \gamma_2, \gamma_3$) into his/her VD_S. **(5)** VD_S concatenates three digits $\gamma_{V_k} = \gamma_1||\gamma_2||\gamma_3$, and encrypts γ_{V_k} with K_{pPO}: $(d_1, e_1) = Enc_{K_{pPO}}(\gamma_{V_k})$. VD_S concatenates both the cipher-texts $(c_1, h_1)||(d_1, e_1)$, and generates NIZKP ($\pi_{enc_{con}}$) to prove that $(c_1, h_1)||(d_1, e_1)$ is equivalent to the concatenation of two ElGamal encrypted cipher-texts under K_{pPO}. The concatenated cipher-texts, PI_{Poll} and t_s are digitally signed by VD_S: $\text{Sign}_{K_{sV_K}}((c_1, h_1)||(d_1, e_1), PI_{Poll}, t_s, \pi_{enc_{con}})$. **(6)** V_k inputs his/her PCS_{ID} into VD_S, who would compute partial codes corresponding to voter's selected v_{ij} options using PRF based on DDH assumption with homomorphic properties. VD_S encrypts each computed partial return code with the public key of CG (K_{pCG}). Also, VD_S generates i NIZKPs (π_{PCS_i}) for each computed partial return code. **(7)** The final ballot ($ballot_{V_k}$) submitted by VD_S to PS consists of the following items:

$$ballot_{V_k} = Enc_{K_{pPS}}(ID_{V_k}), (c_1, h_1)||(d_1, e_1), \pi_{enc}, PI_{Poll}, t_s, \pi_{enc_{con}}, t_{V_k},$$
$$\text{Sign}_{K_{sV_K}}((c_1, h_1)||(d_1, e_1), PI_{Poll}, t_s, \pi_{enc_{con}}), Enc_{K_{pCG}}(F_{\text{DDH}}(PCS_{ID}, v_{1j})),$$
$$Enc_{K_{pCG}}(F_{\text{DDH}}(PCS_{ID}, v_{2j})), Enc_{K_{pCG}}(F_{\text{DDH}}(PCS_{ID}, v_{3j})), \pi_{PCS_1}, \pi_{PCS_2}, \pi_{PCS_3}$$

(where t_{V_k} is the time of voting according to VD_S system clock).

I. Vote Validation: When PS receives $ballot_{V_k}$ from VD$_S$, it starts a verification process. PS decrypts $Enc_{K_{pPS}}(ID_{V_k})$ to obtain ID_{V_k}, and checks if there is already an entry of ballot $ballot_{V_k}$ for V_k. If found, then PS checks the value of the flag (FL) that indicates the number of entries of V_k. If $FL = 0$, i.e. $ballot_{V_k}$ is not found against V_k's record, PS continues the validation process by verifying the digital signature and proofs ($\pi_{enc}, \pi_{enc_{con}}$) contained in $ballot_{V_k}$. PS verifies that t_s and PI_{Poll} used in $ballot_{V_k}$ are equal to the ones sent to V_k. If verified, PS creates a new entry for V_k, stores his/her $ballot_{V_k}$, and sets $FL = 1$. In case $FL = 3$, three entries exist for V_k (i.e. the voter has cast his/her vote three times), PS halts the polling process. If PS finds that there is already an entry of V_k and $FL < 3$, it updates t_s and PI_{Poll} in $ballot_{V_k}$, and checks that the new time is more recent than that of an old entry.

II. Ballot Processing: **(1)** After creation or update of V_k's voting record, PS encrypts voter's id (ID_{V_k}) with K_{pCG}, and sends $Enc_{K_{pCG}}(ID_{V_k})$, encrypted partial return codes ($Enc_{K_{pCG}}(F_{\mathrm{DDH}}(PCS_{ID}, v_{ij}))$, and NIZKPs ($\pi_{PCS_1}$, π_{PCS_2}, π_{PCS_3}) to CG. **(2)** Upon receiving encrypted identity and partial return codes, CG decrypts these using K_{sCG} to obtain ID_{V_k} along with the clear-text of partial return codes. **(3)** CG sends partial return codes, and NIZKPs to each $PCG_{\mathbb{X}}$. **(4)** Each $PCG_{\mathbb{X}}$ verifies NIZKPs, and upon successful verification, computes full return codes using the keyed-PRF and a symmetric key (the same key used to compute the return codes during pre-polling generation of PCS):

$$NRC_{ij}(PCG_{\mathbb{X}}) = hmac(F_{\mathrm{DDH}}(PCS_{ID}, v_{ij}), K_{ses_{\mathbb{X}}}).$$

Each $PCG_{\mathbb{X}}$ encrypts $NRC_{ij}(PCG_{\mathbb{X}})$ with K_{pPCG} to obtain $eNRC_{ij}(PCG_{\mathbb{X}})$:

$$eNRC_{ij}(PCG_{\mathbb{X}}) = Enc_{K_{pPCG}}(NRC_{ij}(PCG_{\mathbb{X}})),$$

$PCG_{\mathbb{X}}$ sends $eNRC_{ij}(PCG_{\mathbb{X}})$ to CG. **(5)** CG computes long return codes NLC_{ij}: $NLC_{ij} = \prod_{\mathbb{X}=1}^{6} eNRC_{ij}(PCG_{\mathbb{X}})$, and sends these long codes to each $PCG_{\mathbb{X}}$. **(6)** Upon receiving NLC_{ij}, each $PCG_{\mathbb{X}}$ looks into its stored mapping table (sent by PF during pre-polling PCS generation) to extract the corresponding short return codes. Each $PCG_{\mathbb{X}}$ encrypts these codes with NLC_{ij}, and sends to CG. **(7)** Upon receiving the encrypted codes, CG uses its stored mapping table (shared with $PCG_{\mathbb{X}}$) to extract the corresponding short return codes ($\widetilde{SC_{ij}}(PCG_{\mathbb{X}})$). Once matching entries are found, CG encrypts the corresponding $\widetilde{SC_{ij}}(PCG_{\mathbb{X}})$ with K_{pPS}, and sends these to PS. **(8)** PS decrypts $Enc_{K_{pPS}}(\widetilde{SC_{ij}}(PCG_{\mathbb{X}}))$ and sends the plain-text short codes to the voter either via mobile connection or an email (in a form of self-destructing message). **(9)** When V_k receives the message from PS, he/she opens it in his/her validation device, and checks whether the received short codes corresponds to the printed short return codes in the PCS. If all the received codes match with the printed ones, V_k inputs ACK to his/her voting device to finalize the ballot casting phase. **(10)** VD_S encrypts ACK with K_{pCG} and sends $Enc_{K_{pCG}}(ACK)$ to PS.

(a) PS sends $Enc_{K_{pCG}}(ACK)$ to CG, who decrypts it with K_{sPS}, and then performs a check on it to confirm that the received ACK is a valid opening for the $Commit_{ACK_i}$. If yes, CG checks the index of $Commit_{ACK_i}$ since there are three published commitments for each voter. CG checks the number corresponding to index "i" of $Commit_{Confirm_i}$. CG extracts the corresponding $Confirm$ code and encrypts it with K_{pPS} and sends it to PS.

(b) PS decrypts the encrypted code and sends $Confirm$ code to V_k. PS adds the $ballot_{V_k}$ to its ballot box. Only the validated votes with "confirmed" codes would be considered in tallying phase. If $Confirm$ code as displayed by V_k's validation device matches with the $Confirm$ code on his/her PCS, the vote confirmation must have been successful. After the confirmation phase, PS generates hash of ballot and publishes it on the BB (concealed from the voters until the final results are announced).

C. Post-polling Phase: After the polling period (t_p) expires, PS no longer accepts the votes. PS sends the list of cipher-texts $\mathscr{C}_k = (c_k, h_k)||(d_k, e_k)$ (stored in its ballot box) with "Confirmed" status to the main PO. The main PO initiates the mixing process to anonymize the votes such that it is impossible to trace which cipher-text belongs to which voter. To mix $\mathscr{C}_0$, a verifiable mixnet is instantiated based on ElGamal encryption, and the shuffle size (equal to the number of POs, i.e. t). Permutation ($\bar{\mathfrak{r}}$) and re-encryption randomizations ($\mathfrak{s}$) are selected at random. To provide the proof of correctness of the mixing, each of these values ($\bar{\mathfrak{r}}$ and $\mathfrak{s}$) must be generated explicitly. Each PO acts like a mixer, who permutes and re-encrypts the ballots ($\mathscr{C}_k^t = (c_k, h_k)||(d_k, e_k)$) and forwards it to the next mixer (PO). It is necessary to protect the privacy of the vote, as the joint decryption phase will reveal vote contents to allow tallying.

I. Mixing and Tallying Phase: $\mathscr{C}_k^1$ is input to the first PO_1 that chooses a random permutation $\bar{\mathfrak{r}}^{(t_1)}$ and permutes the input list to achieve a new list $\overline{\mathscr{C}_k^1} = \left\{(c^1_{\bar{\mathfrak{r}}^{(t_1)}{}_k}, h^1_{\bar{\mathfrak{r}}^{(t_1)}{}_k})||(d^1_{\bar{\mathfrak{r}}^{(t_1)}{}_k}, e^1_{\bar{\mathfrak{r}}^{(t_1)}{}_k})\right\}$. $\bar{\mathfrak{r}}$ only changes the order of the cipher-texts contained in $\mathscr{C}_k$, while the message hidden in the cipher-text remains unchanged. PO_1 re-encrypts ($\overline{\mathscr{C}_k^1}$) using $\mathfrak{s}^1$ to obtain $\mathscr{C}_k^{'1}$. PO_1 submits the mixing result along with the proof (π_{mix_1}) to the BB. BB verifies π_{mix_1} and, on successful verification, posts the mixed vote list for the next PO to mix. $\mathscr{C}_k^{'1}$ is input to the next PO_2, and so on. The output of the last PO_t is the output of the mixing phase. Once all POs have completed the mix, and BB has verified all the proofs ($\pi_{mix_1}, \ldots, \pi_{mix_t}$), the mixing phase is over. The result is an anonymized list of mixed cipher-texts that can be downloaded from BB by each PO to perform decryption using his/her share of secret key, and produce a list of plain-text ballots ($\mathscr{B}_{V_k}||(\gamma_{V_k})$. Each PO must generate and publish NIZKP (π_{dec_t}) on the BB. BB verifies all π_{dec_t} proofs. Once all proofs are validated by BB, the main PO outputs the factors v_{ij} from $\mathscr{B}_{V_k}$ by performing prime factorization. PO checks for each factor to obtain the corresponding voting option, and publishes the output and associated γ_{V_k} on the BB against each polling question.

4 VSPReP Analysis

This section provides an analysis of VSPReP in terms of security and performance.

4.1 Security Analysis

This section discusses the security of VSPReP according to the design requirements and the threat model presented in Sects. 3.2 and 3.3.

A. Coercion Resistance: VSPReP minimizes the possibility of coercion since it allows multiple voting within t_p (with only the last vote being considered valid), and provides multiple PCSs to the authenticated voters. Thus, the voters can always update their votes by using a different PCS, and embedding a new time stamp and a constant PI_{Poll} in the updated ballot before the poll is closed. Therefore, the coercer has no way of knowing if the vote cast in his/her presence and the return codes shown to him/her represents the ballot that was actually counted for that voter. Alternatively, if the coercer has control over the voter's voting device, the voter can forge the contents of the PCS received in the validation device and generate a fake ACK to send to the PS via a controlled voting device. On receiving incorrect ACK, CG would not send a confirmation to the voter, and thus, the ballot would not be considered confirmed and not counted in the tally.

B. Double Vote Prevention: A poll-specific pseudonym PI_{Poll} (signed by PS) is used to identify different votes ($\leq$3) cast by a single authenticated voter during the polling phase to prevent double voting. During vote validation phase, once the PS verifies ID_{V_k}, Sig, and NIZKPs, it checks the received ballot to verify that PI_{Poll} embedded in the ballot matches with the one sent to the voter earlier (step 1 of the polling phase). Since the voter is allowed to vote three times within t_p, the valid ballot must always contain the same PI_{Poll} as described in assumptions (Sect. 3.2). If all other credentials (ID_{V_k}, Sig, and NIZKPs) are verified but PI_{Poll} is not matched, PS halts the polling process. In another possible scenario, a malicious voter may attempt double vote casting by using different identity (pseudo-identity issued at the time of registration). This attack is not possible due to the fact that during three rounds of polling, the pseudo ID of the voter must remain constant. In case of a new pseudo ID request, all the previous votes of the voter shall be revoked by PS on CI's request.

C. Verifiability: Individual verifiability is achieved through the proposed cast-as-intended mechanism based on return codes, which enables the detection of a possible malware attack on the voting device, e.g., if a malicious voting device tries to modify the vote contents, and submit the vote on voter's behalf, the return codes sent to the voter by the PS would not match with the voter's intended voting options. PCGs would also detect the manipulated vote by means of NIZKPs, i.e. the partial return codes and their proofs would not be verified. Moreover, the malicious voting device could not get any information about the received return codes, since the voter uses the validation device to read the message (containing the return codes) received from the PS. In case of mismatch, the voter will then cast his/her vote using a different voting device.

In VSPReP, POs publish the output of the mixing and tallying phase (voting options along with three-digit random codes, associated NIZKPs, and hashes of the confirmed ballots) on the BB so that a voter, any other participant, or auditor can check whether the votes are counted correctly or not. The voters can verify the votes by generating hashes of their submitted ballots, and then compare them to the ones displaying on the BB. Moreover, the published three-digit random code (only known by the voter) on the BB confirms to the voter that his/her vote has been recorded correctly.

D. Privacy of Votes: The possible attacks against the privacy of the votes, as described in Sect. 3.3, can be circumvented in the following ways: (1) a possible coalition between VD_S and PS could not affect vote's privacy, due to the fact that even if a malicious PS verifies incorrect NIZKPs corresponding to manipulated encrypted votes, at the next stage, PCGs would detect the manipulated vote by means of NIZKPs and a voter would not receive any return codes; (2) given the fact that the relation between return codes and the voting options is only known to the voter, neither the PS nor CG/PCGs can use the generated return codes to infer the voter's selected voting options; and (3) after sending the confirmation code to the voter, PS may attempt to collude with CG/PCG to replace the confirmed vote with the colluded vote, i.e. by only replacing the encrypted voting options with their chosen options, and partial return codes computed by brute forcing. However, this attack is infeasible due to the fact PCS_{ID} used by the voter is only known to him/her. Also, at the end of post-polling phase, the voter could compute the hash of the published vote on the BB, and in case of mismatch, complain to the POs of vote manipulation.

4.2 Computational and Cryptographic Costs

We have implemented the polling phase (Sect. 3.4) of VSPReP in Java programming language on a workstation equipped with an Intel i-5 processor at 2.5 GHz and 8 GB of RAM to compute the costs of involved cryptographic operations. Table 1 presents the cryptographic primitives used in the polling phase and the computational costs associated with each operation. The results in Table 1 correspond to 100 runs of each operation on the system (assuming the voter has only cast his/her vote once during t_p). Considering the costs of other operations (computing safe primes, ElGamal key distribution, RSA keys generation, poll setup), on average, a voter requires less than 45 s to cast his/her vote, thus, demonstrating the practicality of the proposed polling protocol.

Table 1. Computational costs of cryptographic primitives.

Phase	Entity	Operations	Time (ms)
VD_S joins with PS	VD_S	TLS	1100
Polling Phase	VD_S	ElGamal Enc of votes and a random no	492
	$VD_S + PS + CG$	RSA Enc/Dec of voter ID	5/38
	$VD_S + CG$	Partial return codes Gen	59
	VD_S	RSA Enc/Dec of partial return codes	11/98
	VD_S	RSA Sig on ballot contents	65
	VD_S	NIZKPs Gen	58
	PS	RSA Sig/NIZKPs Ver	4/545
	PCG	Full return codes Gen	10
	PCG	ElGamal Enc of long and short codes	1895
	$CG + PS$	RSA Enc/Dec of short codes	2/19
	$CG + VD_S$	RSA Enc/Dec of ACK	1/11
	$CG + VD_S$	RSA Enc/Dec of $Confirm$	1/9

In the polling phase, the product of selected voting options, a 3-digit random number, and the long and short return codes are encrypted with ElGamal encryption algorithm, which requires 2 exponentiations each (total 16 modular exponentiations). The voter's pseudo ID, the partial and short return codes, ACK, and $Confirm$ codes are encrypted with the RSA encryption algorithm that requires 1 exponentiation to generate a cipher-text (total 9 exponentiations), and 1 exponentiation to decrypt the cipher-text (total 9 exponentiations). The computation of partial return codes requires 1 exponentiation and M-modular multiplications of each voter selection (1 option per 3 questions that sums up to 3 exponentiations) to polling code sheet ID (a voter-specific key). Two NIZKPs are computed by VD_S: (1) Schnorr identification protocol; and (2) Chaum-Pederson protocol. The generation of first proof requires one modular exponentiation (total 2 for generating ElGamal ciphers) and its verification requires 2 exponentiations (total 4). The second proof requires 2 modular exponentiations (total 6 exponentiations for 3 partial return codes) and its verification requires 4 exponentiations (total 12 exponentiations). The ballot contents are digitally signed using the RSA algorithm that requires one modular exponentiation for signature generation and one modular exponentiation for signature verification. It can be observed that VD_S does not need to perform most expensive cryptographic operations (NIZKP Ver, ElGamal Enc of long and short return codes), which demonstrates the feasibility of implementation of the polling phase on the smartphones.

5 Conclusion

This paper presents a remote polling system, VSPReP, which provides vote anonymity, poll integrity and uncoercibility, and prevents malware infected device to cast a vote on behalf of an authenticated voter during polling phase. VSPReP provides verifiability based on short return codes, a separate voting device, and a BB. To provide cast-as-intended verifiability, VSPReP employs cryptographic primitives to design a complex voting interaction between the voting device and the polling server, which is experimentally shown to be computationally feasible for implementation on portable communication devices. Also, VSPReP supports multiple voting by providing multiple voting sheets, while preventing double voting. As a future work, we intend to address authentication in VSPReP, and develop a working prototype.

Acknowledgment. This work was partly funded by the INCIBEC-2015-02491 "Ayudas para la excelencia de los equipos de investigacin avanzada en ciberseguridad" and TIN2014-57364-C2-2-R "SMARTGLACIS".

References

1. Adida, B.: Helios: web-based open-audit voting. In: SS 2008, pp. 335–348 (2008)
2. Allepuz, J.P., Castelló, S.G.: Cast-as intended verification in Norway. In: EVOTE 2012, pp. 49–63 (2012)

3. Allepuz, J.P., Castelló, S.G.: Internet voting system with cast as intended verification. In: Kiayias, A., Lipmaa, H. (eds.) Vote-ID 2011. LNCS, vol. 7187, pp. 36–52. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32747-6_3
4. Böck, J.: RSA-PSS provable secure RSA signatures and their implementation (2011). https://rsapss.hboeck.de/rsapss.pdf
5. Benaloh, J., Rivest, R., Ryan, P.Y.A., Stark, P., Teague, V., Vora, P.: End-to-end verifiability (2013). https://www.microsoft.com/en-us/research/publication/end-end-verifiablity/
6. Brelle, A., Truderung, T.: Cast-as-intended mechanism with return codes based on PETs. In: Krimmer, R., Volkamer, M., Braun Binder, N., Kersting, N., Pereira, O., Schürmann, C. (eds.) E-Vote-ID 2017. LNCS, vol. 10615, pp. 264–279. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68687-5_16
7. Chaum, D., Pedersen, T.P.: Wallet databases with observers. In: Brickell, E.F. (ed.) CRYPTO 1992. LNCS, vol. 740, pp. 89–105. Springer, Heidelberg (1993). https://doi.org/10.1007/3-540-48071-4_7
8. Fiat, A., Shamir, A.: How to prove yourself: practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (1987). https://doi.org/10.1007/3-540-47721-7_12
9. Galindo, D., Guasch, S., Puiggalí, J.: 2015 Neuchâtel's cast-as-intended verification mechanism. In: Haenni, R., Koenig, R.E., Wikström, D. (eds.) VOTELID 2015. LNCS, vol. 9269, pp. 3–18. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-22270-7_1
10. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Secure distributed key generation for discrete-log based cryptosystems. J. Cryptol. **20**(1), 51–83 (2007)
11. Gjøsteen, K.: The Norwegian internet voting protocol. In: Kiayias, A., Lipmaa, H. (eds.) Vote-ID 2011. LNCS, vol. 7187, pp. 1–18. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32747-6_1
12. Joaquim, R., Ribeiro, C., Ferreira, P.: VeryVote: a voter verifiable code voting system. In: Ryan, P.Y.A., Schoenmakers, B. (eds.) Vote-ID 2009. LNCS, vol. 5767, pp. 106–121. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04135-8_7
13. Krawczyk, H., Bellare, M., Canetti, R.: HMAC: keyed-hashing for message authentication (1997). https://tools.ietf.org/html/rfc2104
14. Mulligan, G.: Has the time now come for internet voting? (2017). http://www.bbc.com/news/business-39955468
15. Naor, M., Pinkas, B., Reingold, O.: Distributed pseudo-random functions and KDCs. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 327–346. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48910-X_23
16. Qureshi, A., Megías, D., Rifà, H.: Framework for preserving security and privacy in P2P content distribution systems. ESWA **42**(3), 1391–1408 (2015)
17. Ryan, P.Y.A., Bismark, D., Heather, J., Schneider, S., Xia, Z.: Prêt à voter: a voter-verifiable voting system. IEEE Trans. Inf. Forensic Secur. **4**(4), 662–673 (2009)
18. Schneider, A., Meter, C., Hagemeister, P.: Survey on remote electronic voting. CoRR (2017). http://arxiv.org/abs/1702.02798
19. Schnor, C.P.: Efficient signature generation by smart cards. J. Cryptol. **4**(3), 161–174 (1991)
20. Terelius, B., Wikström, D.: Proofs of restricted shuffles. In: Bernstein, D.J., Lange, T. (eds.) AFRICACRYPT 2010. LNCS, vol. 6055, pp. 100–113. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-12678-9_7