Designated cast-as-intended verification and universal proof of vote correctness from Chameleon hashes

Tamara Finogina
UPC, Scytl

***Abstract*—This paper presents an e-voting scheme where a voter can verify that her vote has not been changed without requiring to perform any complex computations or rely on pre-delivered information from election authorities. Additionally the scheme allows everyone to verify that all published encrypted votes contain a valid voting option inside. The underlying verification mechanism is inspired by the Challenge-AND-cast method proposed in the work of Guasch and Morillo [7].**

***Index Terms*—electronic voting, cast as intended verifiability, chameleon hash**

I. Introduction

Consider the following situation very common in electronic voting. A voter interacts with a voting system to cast her vote for a preferred voting option and wants to obtain a proof that the option inside the encrypted vote has not been changed by a malicious voting device and was included into a ballot box unmodified i.e. she wants cast-as-intended (CAI) and recorded-as-cast verification. Additionally, it is desirable, that an honest voter cannot prove how she voted after voting phase is over i.e. system should be receipt-free.

There are different ways to enable CAI verification: choice return codes[4], ballot tracking numbers[11], Quick Response (QR) codes storing the encryption randomness[8], cast-or-challenge mechanism[1], etc. Most of the approaches either require voters to perform complex computations in order to validate proofs extracted from a voting device or rely on some secret information delivered to them prior to the election. The former forces voters to use a separate device to handle computations, which according to the research [9] might discourage voters from verifying their votes. The latter means existence of untappable channels between election authorities and each voter, since the delivered information should not be known to anyone but the voter.

An alternative would be to interact with a voter in order to convince her that her intent has not been changed. In such case, only voter herself would be convinced by the CAI proof since only she witnessed the interaction. Moreover, if voting device will additionally simulate proofs for all non-selected options, the proof verification can be outsourced to a public verifier without breaking the voter privacy.

This work presents a universal OR and designated pre-image zero-knowledge proof system (section III), that was inspired by the Challenge-AND-cast technique presented by Guasch and Morillo in [7]. Our method allows voting device to generate OR proof showing to everyone that the encrypted vote contains one of the valid voting options and simultaneously convincing the voter and only the voter, that the vote encrypts her intended option. The sketch of e-voting system with CAI verification based on this method is in section IV, the further improvements are discussed in section V.

II. Preliminaries

a) ElGamal Public Key Encryption: ElGamal encryption scheme [3] is an asymmetric key encryption algorithm for public-key cryptography which is based on the discrete logarithm problem. It is composed of the following algorithms.

Gen: The key generation protocol takes as input a prime number q and a generator g of a cyclic group $\mathbb{G}$ of order q, samples a secret key $\mathtt{sk} \xleftarrow{\$} \mathbb{Z}_q$, computes the matching public key is $\mathtt{pk} = g^{\mathtt{sk}}$ and outputs key pair $(\mathtt{pk}, \mathtt{sk})$.

Enc: The encryption protocol takes as input a public key $\mathtt{pk}$ and a message $m \in \mathbb{G}$, then sample a random value $r \xleftarrow{\$} \mathbb{Z}_q$ and outputs the ciphertext $C = (c_1, c_2)$, where $c_1 = g^r$ and $c_2 = m \cdot \mathtt{pk}^r$.

Dec: The decryption protocol takes as input the secret key $\mathtt{sk}$ and a ciphertext $C = (c_1, c_2) \in \mathbb{G} \times \mathbb{G}$, and outputs $c_2 / c_1^{\mathtt{sk}} = m$.

b) Chameleon Hash: The chameleon hash function scheme [10] is a trapdoor collision-resistant function that behaves as an ordinary collision-resistant function whenever the trapdoor is not used. However, with the trapdoor, collisions can be found easily. We use a chameleon hash function scheme based on the discrete logarithm problem from [7] that is composed by the following algorithms:

$\mathtt{Gen}_{ch}$: takes as input group parameters (p, q, g), samples a trapdoor key $x \xleftarrow{\$} \mathbb{Z}_q$, computes an evaluation key $h = g^x$ and outputs chameleon hash scheme keys (h, x).

$\mathtt{H}_{ch}$: takes as input an evaluation key h, a message $m \in \mathbb{Z}_q$ and a random $\psi \in \mathbb{Z}_q$ and outputs a commitment $c_{ch} = g^m h^\psi \in \mathbb{G}$.

$\mathtt{H}_{ch}^{-1}$: takes as input the trapdoor key x, two messages $m, m' \in \mathbb{Z}_q$ and a random $\psi \in \mathbb{Z}_q$ and returns a value $\psi' = (m - m')x^{-1} + \psi \in \mathbb{Z}_q$.

Chameleon hashes have the following properties:

Collision resistance: If given only h, $Pr[(m, \psi) \neq (m', \psi') : \mathcal{H}_{ch}(h, m, \psi) = \mathcal{H}_{ch}(h, m', \psi')] \approx 0$.

Uniformity: $\forall m \in \mathbb{Z}_q$ and $\forall \psi$ uniformly distributed in $\mathbb{Z}_q$, c_{ch} is uniformly distributed in $\mathbb{G}$.

c) Interactive zero-knowledge proof systems: Let us consider two probabilistic polynomial time (PPT) algorithms, the prover $\mathtt{P}$ and the verifier $\mathtt{V}$, and let a prover $\mathtt{P}$ to have some secret witness w of the fact that some public element x

belongs to some language $\mathcal{L}_{\mathcal{R}}$, where $\mathcal{R} \subset \mathcal{X} \times \mathcal{W}$ is a binary relation and $\mathcal{L}_{\mathcal{R}} = \{x \in \mathcal{X} \mid \exists w \in \mathcal{W}\ s.t. (x, w) \in \mathcal{R}\}$. In an interactive zero-knowledge proof system, the prover P and a verifier V interact, and the goal of this interaction is for P to convince V that it knows a secret witness w such that $x \in \mathcal{L}_{\mathcal{R}}$. Typically, three properties are required for such a protocol [5]:

- **Completeness:** if both P and V are honest and $(x, w) \in \mathcal{R}$, then the verifier always accepts the proof as valid.
- **Soundness:** if P is dishonest and $(x, w) \notin \mathcal{R}$, then the verifier does not accept the proof as valid.
- **Zero-Knowledge:** the execution of the protocol does not leak any information about the secret witness w. This is formalized requiring, for every verifier $\mathtt{V}^*$, the existence of a polynomial-time simulator $\mathtt{M}_{\mathtt{V}^*}$ such that for every $(x, w) \in \mathcal{R}$ the output $\langle \mathtt{P}(x, w), \mathtt{V}^*(x) \rangle$ is identically distributed to the output $\mathtt{M}_{\mathtt{V}^*}(x)$.

d) Sigma protocols: Let P, V and $\mathcal{R}$ be as above. A three-move interactive protocol $\langle \mathtt{P}, \mathtt{V} \rangle$ with transcript (a, e, z), where the first message a is sent by the prover and e is a randomly distributed value drawn from a suitable challenge space of size t, it is a Σ-*protocol* for $\mathcal{R}$ if it is complete (in the above sense) and the following holds [2],[6]:

Special soundness: There exists a PPT algorithm K, such that from any $x \in \mathcal{L}_{\mathcal{R}}$, and any pair of accepting transcripts (a, e, z), (a, e', z') it holds $Pr[(x, w) \in \mathcal{R} \mid w \leftarrow K(x, (a, e, z), (a, e', z')] \geq 1 - negl(t)$.

Special honest-verifier zero-knolwedge: There exists a PPT algorithm $\mathtt{M}_{\mathcal{R}}$, which on input x and a random e, outputs an accepting transcript (a, e, z) with the same probability distribution as $\langle \mathtt{P}, \mathtt{V} \rangle$.

Example: Proving in Zero-Knowledge that C is an ElGamal encryption of m.

Suppose the prover P has encrypted the message m, using ElGamal and randomness $r \in \mathbb{Z}_q$, and the result is the ciphertext $c = (c_1, c_2) = (g^r, m \cdot \mathtt{pk}^r)$. To convince V that c is indeed encryption of m, P and V will engage in a zero-knowledge proof of the equality of two discrete logarithms that works as follows:

1) P samples $s \xleftarrow{\$} \mathbb{Z}_q$, then sends as the first message the commitment $a = (a_1, a_2) = (g^s, \mathtt{pk}^s)$.
2) V chooses uniformly at random a challenge $e \xleftarrow{\$} \mathbb{Z}_q$ and sends it to P.
3) P computes $z = s + e \cdot r \mod q$ and sends it to V.
4) V accepts the proof as valid if the two following equalities hold: $g^z = a_1 \cdot c_1^e$ and $\mathtt{pk}^z = a_2 \cdot \left(\frac{c_2}{m}\right)^e$.

To simulate the proof, a simulator M takes (c, m), samples $z, e \xleftarrow{\$} \mathbb{Z}_q$ uniformly at random, computes $a_1 = g^z \cdot c_1^{-e}$ and $a_2 = \mathtt{pk}^z \cdot \left(\frac{c_2}{m}\right)^{-e}$ and outputs a, e, z, where $a = (a_1, a_2)$.

III. Designated verification of a statement from a set with universal set membership verification

Suppose there are n possible valid voting option $M = (m_1, \ldots, m_n)$. The voter wants to make sure that an ElGamal encryption $c = (c_1, c_2) = (g^r; m_i \cdot \mathtt{pk}^r)$ contains her intent m_i, while everyone else wants to verify that c contains *some* option from the set M i.e. c is an encryption of a valid option. Such verification provides CAI verification to the voter and proof of vote correctness to everyone.

Formally, it means constructing designated proof for the index i known to V that an i-th statement $x_i = (c_1, \frac{c_2}{m_i})$ of the set $X = \left((c_1, \frac{c_2}{m_1}), \ldots, (c_1, \frac{c_2}{m_n})\right)$ belongs to $\mathcal{L}_{\mathcal{R}}$ and an OR proof that $X \in \mathcal{L}_{\mathtt{OR}}$, where $\mathcal{L}_{\mathtt{OR}} = \{(x_1, \ldots, x_n), w| \ \exists i \text{ s.t. } x_i \in \mathcal{L}_{\mathcal{R}}\}$ and $\mathcal{L}_{\mathcal{R}} = \{(x, w) \mid x = (g^w; \mathtt{pk}^w)\}$. Note that due to Decisional Diffie-Hellman (DDH) assumption it is hard to decide in polynomial time if a statement from set X belongs to $\mathcal{L}_{\mathcal{R}}$ or not.

To construct designated verification combined with OR proof, P generates non-interactive fake proofs for all $\forall x_j \in X \setminus x_i$ s.t. $x_j \notin \mathcal{L}_{\mathcal{R}}$ and starts an interactive real proof generation for $x_i \in \mathcal{L}_{\mathcal{R}}$. To generate a real proof π, P executes the first step of the Σ-protocol to get a commitment H to the real proof, then engages in an interaction with V to obtain a random challenge γ according to which the real proof will be modified $\hat{\pi}$. The modification is such that only real proof can pass verification after the modification. After the interaction, P outputs $n - 1$ fake proofs $\{\pi_j^*\}$ and a modified real proof $\hat{\pi}$. To avoid a trivial correlation, all proofs are output in the indices-based order. The workflow is presented in Figure 1.

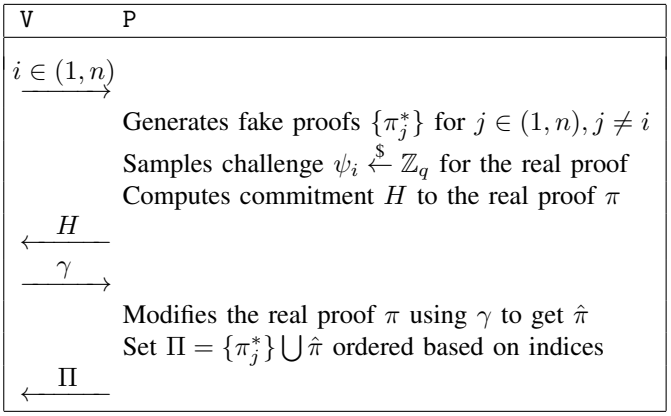

Figure 1. Interaction between P and V.

To fake proofs for non instances of $\mathcal{L}_{\mathcal{R}}$, P will use a simulator for an interactive Σ-protocol $\mathtt{M}_{\mathtt{V}^*}$ and trapdoor collision of a chameleon hash function. If $\mathcal{L}_{\mathcal{R}}$ is a hard-to-distinguish language inside $\mathcal{X}$, thus the simulator $\mathtt{M}_{\mathtt{V}^*}$ must work also when the input element x_j is not in the language, that is $x_j \in \mathcal{X} \setminus \mathcal{L}_{\mathcal{R}}$. Furthermore, any polynomial-time distinguisher will fail in distinguishing the joint distributions $(x_i, \mathtt{M}_{\mathtt{V}^*}(x_i))_{x_i \leftarrow \mathcal{L}_{\mathcal{R}}}$ and $(x_j, \mathtt{M}_{\mathtt{V}^*}(x_j))_{x_j \leftarrow \mathcal{X} \setminus \mathcal{L}_{\mathcal{R}}}$; if this was not the case, such a distinguisher could be used to distinguish the language $\mathcal{L}_{\mathcal{R}}$ inside $\mathcal{X}$.

Remaining section organization is the following: first part (Section III-A) shows a designated pre-image zero-knowledge proof system based on chameleon hashes, the second (Section III-B) proves that the proposed scheme is correct, sound and honest-verifier zero knowledge, while the third (Section III-C) shows how it can be applied to a set of statements to obtain designated verification combined with universal membership verification.

A. Designated pre-image zero-knowledge proof system based on chameleon hashes

All algorithms described in this section take as implicit input a common reference string $\mathtt{crs} = (q, g, \mathbb{G}, \mathtt{pk}, M, \mathtt{aux})$,

where g is a generator of a cyclic group $\mathbb{G}$ of a prime order q, $p = 2q + 1$ is a safe prime, $\mathtt{pk} \in \mathbb{G}$ is a public ElGamal encryption key, $M = (m_1, \ldots, m_n)$ is a set of valid voting options and `aux` is an auxiliary information.

Additionally we require two collision-resistant hash functions $\mathtt{Hash}_1 : \{0,1\}^* \to \mathbb{Z}_q$ and $\mathtt{Hash}_2 : \{0,1\}^* \to \mathcal{M}$ (the space of t-bits values that voter can remember). The second hash function is necessary for the usability as a voter cannot be expected to remember large strings of random data.

The scheme consist of the seven algorithms `GenKeys`, `NIZKProveOriginal`, `ZKProveModified`, `ZKSimulate`, `GetCommitment`, `ZKVerify`, `ZKVerifyDesignated` defined as follows:

`GenKeys`: runs $\mathtt{Gen}_{ch}(q, g, \mathbb{G})$ and outputs chameleon hash scheme keys (h, x).

`NIZKProveOriginal`: receives as input an ElGamal ciphertext $c = (c_1, c_2)$, a true selection $m_i \in M$ s.t. $(c_1, \frac{c_2}{m_i}) \in \mathcal{L}_{\mathcal{R}}$, the trapdoor key x and a chameleon challenge ψ for the real proof. Then it does the following:

Samples a random element $s \xleftarrow{\$} \mathbb{Z}_q$
$a \leftarrow (g^s, \mathtt{pk}^s)$.
$H \leftarrow \mathtt{Hash}_2(c_1, \frac{c_2}{m_i}, a, \psi, x, \mathtt{crs})$.
Outputs the first step of the Σ-protocol (a, s) and the commitment to the real proof H.

`ZKProveModified`: receives as input an ElGamal ciphertext $c = (c_1, c_2)$, a true selection $m_i \in M$ s.t. $(c_1, \frac{c_2}{m_i}) \in \mathcal{L}_{\mathcal{R}}$, the values (a, s) and the chameleon challenge ψ for the real proof, the randomness r used for the ciphertext, a challenge γ and the chameleon hash keys (h, x). Then it does the following:

$\hat{a} \leftarrow a^{\gamma}$.
$\hat{e} \leftarrow \mathtt{Hash}_1\left(\mathtt{H}_{ch}\big(h, \mathtt{Hash}_1(c_1, \frac{c_2}{m_i}, \hat{a}, x, \mathtt{crs}), \psi\big)\right)$
$\hat{z} \leftarrow s\gamma + \hat{e}r$.
Outputs the proof $\hat{\pi} = (\hat{a}, \hat{e}, \hat{z}, \psi)$

`ZKSimulate`: takes as input an ElGamal ciphertext $c = (c_1, c_2)$, a message $m_j \in M$ s.t. $(c_1, \frac{c_2}{m_j}) \notin \mathcal{L}_{\mathcal{R}}$, the chameleon hash scheme keys (h, x). Then it does the following:

Sample random elements $z^*, \alpha, \beta \xleftarrow{\$} \mathbb{Z}_q$.
$e^* \leftarrow \mathtt{Hash}_1\big(\mathtt{H}_{ch}(h, \alpha, \beta)\big)$
$a^* \leftarrow (g^{z^*} c_1^{-e^*}, \mathtt{pk}^{z^*}(\frac{c_2}{m_j})^{-e^*})$.
$\psi^* \leftarrow \mathtt{H}_{ch}^{-1}(x, \alpha, \mathtt{Hash}_1(c_1, \frac{c_2}{m_j}, a^*, x, \mathtt{crs}), \beta)$.
Outputs the proof $\pi^* = (a^*, e^*, z^*, \psi^*)$.

`GetCommitment`: receives as input a proof $\pi = (a, e, z, \psi)$, an ElGamal ciphertext $c = (c_1, c_2)$, a message $m \in M$, a challenge γ, a trapdoor key x and outputs a commitment $H = \mathtt{Hash}_2(c_1, \frac{c_2}{m}, (a)^{\gamma^{-1}}, \psi, x, \mathtt{crs})$.

`ZKVerify`: receives as input proofs $\{\pi_j = (a_j, e_j, z_j, \psi_j,)\}_{j=1}^n$ for all messages in M, commitments to the proofs $\{h_j\}_{j=1}^n$, the ElGamal ciphertext $c = (c_1, c_2)$, a challenge γ and the chameleon hash keys (h, x). Then it returns $\top$ if and only all checks are correct. Otherwise it outputs $\perp$.

Checks $h \stackrel{?}{=} g^x$.
For j from 1 to n:
$e \leftarrow \mathtt{Hash}_1\left(\mathtt{H}_{ch}\big(h, \mathtt{Hash}_1(c_1, \frac{c_2}{m_j}, a, x, \mathtt{crs}), \psi_j\big)\right)$
Checks $e_j \stackrel{?}{=} e$
Checks $(g^{z_j}; \mathtt{pk}^{z_j}) \stackrel{?}{=} (a_{j1} c_1^{e_j}; a_{j2}(\frac{c_2}{m_j})^{e_j})$, where $a_j = (a_{j1}, a_{j2})$
Checks $H \stackrel{?}{=} \mathtt{GetCommitment}(\pi, c, m_j, \gamma, x)$.

`ZKVerifyDesignated`: receives as input the commitment H given by `P` before `V` introduced the challenge, the intended option m_i, the mapping table $\{\gamma||m_j||H_j\}_{j=1}^n$ and the challenge γ^* introduced by `V`. Then it returns $\top$ if and only if all checks are correct. Otherwise it outputs $\perp$.

Finds line $\gamma||m_i||H_i$ that corresponds to m_i.
Checks $\gamma \stackrel{?}{=} \gamma^*$
Checks $H \stackrel{?}{=} H_i$.

B. Security of the proposed zero-knowledge proof system

Theorem 1. *The scheme defined in the Section III is complete if the underlying Σ-protocol is complete and the chameleon hash function fulfills the property of correctness.*

Proof: The completeness of the protocol follows easily from the inspection. □

Theorem 2. *The scheme defined in Section III is sound in the random oracle model if the underlying Σ-protocol has the special soundness property and the challenge γ is a random value.*

Proof: Assume that there exist an adversary $\mathcal{A}$ that is able to produce two proofs $\pi_0 = ((a_{01}, a_{02}), e_0, z_0, \psi_0), \pi_1 = ((a_{11}, a_{12}), e_1, z_1, \psi_1)$ and commitments to these proofs H_0, H_1 for the statement (c, m) for two different challenges γ_0, γ_1 such that commitments are identical i.e. $H_0 = H_1 = H$.

Recall, that a commitment to the original proof is verified as follows:
$H_0 = \mathtt{Hash}_2(c_1, \frac{c_2}{m}, (a_{01}^{\gamma_0^{-1}}, a_{02}^{\gamma_0^{-1}}), \psi_0, x_0, \mathtt{crs})$.
$H_1 = \mathtt{Hash}_2(c_1, \frac{c_2}{m}, (a_{11}^{\gamma_1^{-1}}, a_{12}^{\gamma_1^{-1}}), \psi_1, x_1, \mathtt{crs})$.

Assuming probability of a collision of a hash function $\mathtt{Hash}_2$ is negligible, we have that $H_0 = H_1$ if and only if all inputs are identical. Therefore:
$\psi_0 = \psi_1 = \psi$,
$x_0 = x_1 = x$,
$(a_{01})^{\gamma_0^{-1}} = (a_{11})^{\gamma_1^{-1}} = a_1$,
$(a_{02})^{\gamma_0^{-1}} = (a_{12})^{\gamma_1^{-1}} = a_2$.

Since, both proofs are correct, then verification of the underlying Σ-proof must hold. In other words:
$(g^{z_0}; \mathtt{pk}^{z_0}) = (a_{01} c_1^{e_0}; a_{02}(\frac{c_2}{m})^{e_0}) = (a_1^{\gamma_0} c_1^{e_0}; a_2^{\gamma_0}(\frac{c_2}{m})^{e_0})$
$(g^{z_1}; \mathtt{pk}^{z_1}) = (a_{11} c_1^{e_1}; a_{12}(\frac{c_2}{m})^{e_1}) = (a_1^{\gamma_1} c_1^{e_1}; a_2^{\gamma_1}(\frac{c_2}{m})^{e_1})$.
Thus, either (a_1, a_2) is a commitment of a real Σ-protocol or there is a simulator $\mathcal{S}^*$ that generates $a^* = (a_1^*, a_2^*)$ such that for any γ there are z^*, e^* such that $(g^{z^*}; \mathtt{pk}^{z^*}) = (a_1^*)^{\gamma} c_1^{e^*}; (a_2^*)^{\gamma}(\frac{c_2}{m})^{e^*})$. However, such $\mathcal{S}^*$ can be used to break the special soundness property of a Σ protocol. Therefore, (a_1, a_2) is a commitment to a real Σ-protocol and the witness r is extractable: $z_0\gamma_0^{-1} - z_1\gamma_1^{-1} = s\gamma_0\gamma_0^{-1} + re_0\gamma_0^{-1} - s\gamma_1\gamma_1^{-1} - re_1\gamma_1^{-1} = r(e_0\gamma_0^{-1} - e_1\gamma_1^{-1})$. Therefore, $\frac{z_0\gamma_0^{-1} - z_1\gamma_1^{-1}}{e_0\gamma_0^{-1} - e_1\gamma_1^{-1}} = r$ □

Theorem 3. *The scheme defined in Section III is honest-zero*

knowledge in the common reference string model if the Σ*-protocol is honest-zero knowledge verifier.*

Proof: Consider two proofs $\hat{\pi} = (\hat{a}, \hat{e}, \hat{z}, \hat{\psi})$ and $\pi^* = (a^*, e^*, z^*, \psi^*)$ generated by `ZKProveModified` and `ZKSimulate` respectively.

From the zero-knowledge property of the Σ-protocol, we have that (a, z) of the real Σ-protocol execution and (a^*, z^*) produced by a simulator are indistinguishable. Recall, that in the modified proof $\hat{z} = s\gamma + r\hat{e}$ is generated in the same way as $z = s + re$ in the Σ-protocol but based on $s\gamma$ instead of s and using a chameleon hash challenge to compute $\hat{e}$ instead of a normal hash in e computation. Since γ is invertible and not random and s is random, $s\gamma$ is random. Due to the uniformity property of chameleon hashes we get that $\hat{e}$ and e are indistinguishable. Also, a and $\hat{a} = a^{\gamma}$ are indistinguishable since $\mathcal{L}_{\mathcal{R}}$ is hard to distinguish. Therefore $\hat{\pi}$ and π are indistinguishable. □

C. Designated verification combined with universal OR proof

The proposed zero-knowledge proof scheme can be converted to a designated verification of a statement x_i combined with a universal set membership verification i.e. designated $x_i \in \mathcal{L}_{\mathcal{R}}$ and universal $x_i \in X$, where $X = \{(c_1, \frac{c_2}{m_j})\}_{m_j \in M}$. In other words, anyone will be able to check that an ElGamal ciphertext c indeed encrypts a valid voting option, but only `V` will be able to identify which option m_i precisely.

To construct a universal set membership proof, we need to make the challenge ψ_i of the real proof depend on fake proofs. As before, `P` will run `ZKSimulate` for all $x_j = (c_1, \frac{c_2}{m_j}) \in X$ s.t. $m_j \neq m_i$ to obtain a set of simulated proofs $\{\pi_j = (a_j, e_j, z_j, \psi_j, x)\}_{j=1, j\neq i}^{n}$. However, instead of sampling a random ψ_i for the real proof, `P` computes $\psi_i = \mathtt{Hash}_1(c, \mathtt{crs}) - \sum_{j=1, j\neq i}^{n} \psi_j$ and uses it to obtain the commitment to the real statement h. After that, the process goes identical to Fig. 1. The workflow for designated verification combined with universal set membership check is shown in Fig. 2.

V	P
	Runs `GenCRS`
	Runs `ZKSimulate` for $\forall m_j \neq m_i$ to get $\{\pi_j^*\}$
	Sets $\psi_i = \mathtt{Hash}_1(c, \mathtt{crs}) - \sum_{j=1, j\neq i}^{n} \psi_j$
	Runs `NIZKProveOriginal` with ψ_i to get H
$\xleftarrow{H}$	
$\xrightarrow{\gamma}$	
	Runs `ZKProveModified` using γ to get $\hat{\pi}$
	Set $\Pi = \{\pi_j^*\} \bigcup \hat{\pi}$ ordered as set X
$\xleftarrow{\Pi}$	

Figure 2. Interaction between `P` and `V` for achieving designated verification of a statement $x_i = (c_1, \frac{c_2}{m_i})$ along with universal verification of a set membership.

Theorem 4. *Proof* $\Pi = \{\pi^j\} \bigcup \hat{\pi}$ *for the statements in set* $X = \{x^j\} \bigcup x_i$*, where* $\{\pi^j\}$ *are generated by* `ZKSimulate` *and* $\hat{\pi}$ *is an output of* `ZKProveModified` *defined Section III, is a universal OR-proof that* $X \in \mathcal{L}_{\mathtt{OR}}$ *and designated verifier proof that* $x_i \in \mathcal{L}_{\mathcal{R}}$ *for the index* i*, when* $\sum_j \psi_j = \mathtt{Hash}_1(c, \mathtt{crs})$.

Proof: Zero-knowledge proof scheme defined in Section III is sound, therefore it is sufficient to show that Π cannot contain only simulated proofs. For the sake of contradiction, suppose it is possible to construct simulated proofs and ensure that $\sum_j \psi_j = \mathtt{Hash}_1(c, \mathtt{crs})$. This implies that we can construct at least one simulated proof for an option m^* using a fixed value ψ^*. Recall, that `ZKSimulate` uses the trapdoor x to obtain a value ψ^* such that $e^* = \mathtt{Hash}_1\left(\mathtt{H}_{ch}\left(h, \mathtt{Hash}_1(c_1, \frac{c_2}{m^*}, a^*, x, \mathtt{crs}), \psi^*\right)\right)$.

If ψ^* is a fixed value, the equality above has a negligible chance to hold for fixed $c_1, \frac{c_2}{m^*}, \mathtt{crs}$ and e^*, a^* computed by the simulator $\mathcal{S}$. Therefore, to construct a simulated proof for a fixed ψ^* it should be possible to modify at least one of $c_1, \frac{c_2}{m^*}, e^*, a^*$ and pass Σ-protocol and chameleon hash verifications. However, $c_1, \frac{c_2}{m^*}, a^*$ are part of $\mathtt{Hash}_1(c_1, \frac{c_2}{m^*}, a^*, x, \mathtt{crs})$ hash and any strategy to adjust inputs to get a specific hash output implies finding a collision for $\mathtt{Hash}_1$, which is a collision-resistant by assumption. Similarly, if we find pair e^*, a^* that together with ψ^* passes chameleon hash verification, then in order to adjust z^* to pass Σ-protocol verification we need to break soundness of the underlying Σ protocol. □

IV. Cast-as-intended verification

This section sketches CAI verification mechanism based on trapdoor zero-knowledge proof scheme from section III that provides universal assurance of ballot correctness. Note that the sketch focuses only on CAI verification and omits many important steps such as authentication, credential generation, election constitution, tallying etc. Therefore we assume that $\mathtt{crs} = (p, q, g, \mathtt{pk}, M = (m_1, \ldots, m_n), \mathtt{aux})$ is already pre-generated and is known to all parties. Also we assume, that each voter already has a pair of signing keys $(\mathtt{pk}_s, \mathtt{sk}_s)$.

The CAI scheme consists of the following algorithms:

`GenEncAndCommit` takes as input a voting option $m_i \in M$ and does the following:

Samples randomness $r \xleftarrow{\$} \mathbb{Z}_q$
Encrypts the selection $c = (c_1, c_2) \leftarrow Enc_{pk}(m_i; r)$.
Gets the chameleon hash keys $(h, x) \leftarrow \mathtt{GenKeys}()$.
Computes $n-1$ simulated proofs $\{\pi_j\}_{j=1, j\neq i}^{n}$, where $\pi_j = (a_j, e_j, z_j, \psi_j) \leftarrow \mathtt{ZKSimulate}(c, m_j, h, x)$.
Computes $\psi_i \leftarrow \mathtt{Hash}_1(c, \mathtt{crs}) - \sum_{j=1, j\neq i}^{n} \psi_j$.
Starts the real proof generation and commits to it $a, s, H \leftarrow \mathtt{NIZKProveOriginal}(c, m_i, h, x, \psi_i)$.
Outputs $(c, r, a, s, \psi_i, H, h, x, \{\pi_j\}_{j=1, j\neq i}^{n})$.

`FinishProofsGen` takes as input the ciphertext c, the randomness r, the challenge γ and the values $(a, s, \psi_i, h, x, \{\pi_j\}_{j=1, j\neq i}^{n})$ generated by `GenEncAndCommit` algorithm and does the following:

$\pi_i \leftarrow \mathtt{ZKProveModified}(c, m_i, a, s, \psi_i, r, \gamma, h, x)$
Outputs $\{\pi_j\}_{j=1}^{n}$

`GenMappingTable` takes as input the ciphertext c, the challenge γ, the proofs $\{\pi_j\}_{j=1}^{n}$ and chameleon hash keys (h, x). Then it does the following:

Computes commitments $\{H_j\}_{j=1}^{n}$ to all proofs, where $h_j \leftarrow \mathtt{GetCommitment}(\pi_j, c, m_j, \gamma)$.

If $\texttt{ZKVerify}(\{\pi_j, H_j\}_{j=1}^n, c, \gamma, h, x)$ outputs $\perp$, abort. Else, prepares a mapping table $\{(\gamma||m_j||H_j)\}_{j=1}^n$.

The voting scheme consists of four entities: voter, voting device (VD), public ballot box (BB) and auditors. We assume that voter and auditors are trusted, while BB is a passive entity with append only functionality.

The voting and verification of the vote happens as follows:

Voter: Selects her voting option $m_i \in M$ and introduces m_i and her private signing key $\texttt{sk}_s$ into the Voting device.

VD: Runs $\texttt{GenEncAndCommit}(m_i)$ to get values $(c, r, a, s, \psi_i, H, h, x, \{\pi_j\}_{j=1, j\neq i}^n)$. Then shows H to the voter.

Voter: Remembers H and introduces a challenge γ.

VD: Runs `FinishProofsGen` to get set of proofs $\{\pi_j\}_{j=1}^n$. Then it signs proofs and ciphertext with voter's signing key $\texttt{sk}_s$ sends a ciphertext c, proofs $\{\pi_j\}_{j=1}^n$ and the signature to the BB.

BB: Upon receiving $c, \{\pi_j\}_{j=1}^n$ and the signature, verifies that voter did not already vote. If there is a vote in database for that voter, it does nothing. Otherwise, it adds the received values to a database and publishes them.

Auditors: Once a new vote is added to BB, they execute `GenMappingTable` and publish the resulting mapping table or mark the vote as invalid.

Voter: Goes to the public BB, checks that there is a mapping table for her vote. Then she verifies that published challenge γ corresponds to the challenge she used and that commitment for her option m_i is equal to the H voting device showed here before she introduced the challenge. This verification is equivalent to executing $\texttt{ZKVerifyDesignated}(H, m_i, \{(\gamma||m_j||H_j)\}_{j=1}^n, \gamma)$.

Only voter herself can ensure that the ciphertext c indeed contains an encryption of her selection m_i, since only she witnessed interaction and knows which H is correct. For others all proofs are indistinguishable and only reveal that an encrypted value is a valid voting option from set M. Additionally, the scheme is a receipt-free, since an honest voter can name any H as a value her voting device showed her. An adversary will have to believe her, since only she witnessed the interaction.

Therefore, this scheme is receipt-free, supports cast-as-intended and recorded as cast verification for voter and simultaneously provides a universal proof of vote correctness.

V. Future work

The proposed scheme achieves receipt-freeness, however it does not provide coercion-resistance, which is a stronger requirement. A coercer still cannot witness interaction between a voter and her voting device, however it has more powers over voter and can instruct a voter to behave a certain way. For example, a coercer can force the voter to select a specific challenge $\gamma = F(H)$ for some function F. Such choice of γ would allow coercer to ensure that voter obeyed as the probability of a random H^* to satisfy such constraint is negligible.

The problem can be mitigated to some extend by outputting H in a non-numerical form e.g. an image, however it will not solve the problem completely, since malicious or coerced voter might use an additional tool to revert H to the required form.

Alternatively, e-voting application might require voter to use a picture or some bio-metric data as a challenge γ. However such case should be further studied as bio-metric data is typically withing some range and a malicious voting device can use slightly alerted γ without being caught.

Another possibility to address the coercion-resistance would be to show voter a set of possible challenges $\{\gamma_1, \ldots, \gamma_k\}$ and ask her to select one among them. This solution will be coercion-resistant, however it comes with the cost of reduced soundness.

One more approach would be to include an additional possibly malicious entity (Voting Server (VS)) and use hash-chains to set the order of operations. In such a case, the workflow will be slightly modified:

Voter: Introduces $m_i \in M$ into the Voting device.

VD: Runs `GenEncAndCommit`, then shows H to the voter.

Voter: Remembers H and introduces her private signing key $\texttt{sk}_s$.

VD: Signs the vote c and sends c and the signature to the voting server.

VS: Verifies the signature, adds the received values to the BB, computes and outputs a new state $st_{k+1} = \texttt{Hash}(BB, st_k)$, where the st_0 is set to some initial value.

VD: Runs `FinishProofsGen` using st_{k+1} as γ to get proofs $\{\pi_j\}_{j=1}^n$, signs them and sends proofs and the signature to the Voting Server.

VS: Publishes the revived values.

Auditors: One a new vote is published, execute `GenMappingTable` and publishes the resulting mapping table on an auditors' BB or marks the vote as invalid.

Voter: Goes to the auditors' BB and verifies that commitment for her option m_i is equal to the H the voting device showed here before she introduced the signing key.

As we have shown, there are several approaches to address the problem of combining coercion-resistant with CAI verification without relying on trusted channels and additional devices. Nevertheless, this direction should be further studied.

References

[1] Ben Adida. Helios: Web-based open-audit voting. pages 335–348, 01 2008.

[2] Ivan Damgård. On σ-protocols. https://www.cs.au.dk/\~ivan/Sigma.pdf/.

[3] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Inf. Theory*, 31(4):469–472, 1985.

[4] Kristian Gjøsteen. The norwegian internet voting protocol. In Aggelos Kiayias and Helger Lipmaa, editors, *E-Voting and Identity*, pages 1–18, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[5] Oded Goldreich. *Foundations of Cryptography*, volume 1. Cambridge University Press, 2001.

[6] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In Robert Sedgewick, editor, *Proceedings of the 17th Annual ACM Symposium on Theory of Computing, May 6-8, 1985, Providence, Rhode Island, USA*, pages 291–304. ACM, 1985.

[7] Sandra Guasch and Paz Morillo. How to challenge and cast your e-vote. In Jens Grossklags and Bart Preneel, editors, *Financial Cryptography and Data Security*, pages 130–145, Berlin, Heidelberg, 2017. Springer Berlin Heidelberg.

[8] S. Heiberg and J. Willemson. Verifiable internet voting in estonia. In *2014 6th International Conference on Electronic Voting: Verifying the Vote (EVOTE)*, pages 1–8, 2014.

[9] F. Karayumak, M. Kauer, M. M. Olembo, T. Volk, and M. Volkamer. User study of the improved helios voting system interfaces. In *2011 1st Workshop on Socio-Technical Aspects in Security and Trust (STAST)*, pages 37–44, 2011.

[10] Hugo Krawczyk and Tal Rabin. Chameleon signatures. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2000, San Diego, California, USA*. The Internet Society, 2000.

[11] Peter Ryan, Peter Rønne, and Vincenzo Iovino. Selene: Voting with transparent verifiability and coercion-mitigation. volume 9604, pages 176–192, 02 2016.