Anamorphic Voting: Ballot Freedom Against Dishonest Authorities

Rosario Giustolisi[1], Mohammadamin Rakeei[2], and Gabriele Lenzini[2]

[1] IT University of Copenhagen, Copenhagen, Denmark
rosg@itu.dk
[2] SnT, University of Luxembourg, Esch-sur-Alzette, Luxembourg
{amin.rakeei,gabriele.lenzini}@uni.lu

Abstract. Electronic voting schemes typically ensure ballot privacy by assuming that the decryption key is distributed among tallying authorities, preventing any single authority from decrypting a voter's ballot. However, this assumption may fail in a fully dishonest environment where all tallying authorities collude to break ballot privacy.
In this work, we introduce the notion of *anamorphic voting*, which enables voters to convey their true voting intention to an auditor while casting an (apparently) regular ballot. We present new cryptographic techniques demonstrating that several existing voting schemes can support anamorphic voting.

1 Introduction

According to the Democracy Index published by the Economist Group [28] in 2024, 59 countries are classified as authoritarian regimes. To remain in power, such regimes need to demonstrate overwhelming support for the ruling party, often designing voting systems to ensure that the dictator decisively wins the election. Therefore, elections in these countries often lack transparency, raising concerns about the secrecy and integrity of the ballots.

An authoritarian regime that (falsely) seeks to address such concerns could still implement a voting scheme that is proven to be secure, while violating its trust assumptions. For example, ballot privacy is clearly compromised in most voting systems if an attacker controls all the election decryption keys and has access to the datasets containing the ballots cast at the voting phase. The absence of vote privacy intimidates voters, preventing them from freely expressing their true preferences. However, is it still possible for voters to communicate their real voting intentions without being detected by the authoritarian regime?

In this work, we introduce *anamorphic voting*, exploring how participants in an election might communicate covertly even when an authoritarian regime controls *all* communication channels and implements an *existing* voting scheme without satisfying the necessary trust assumptions. Anamorphic voting is inspired by the recent concept of anamorphic encryption [25] in the context of electronic voting. While anamorphic encryption studies whether *a single* well-established public-key cryptosystem supports covert communication under stringent dictator conditions, anamorphic voting examines whether *a combination* of

cryptographic primitives within a voting scheme can enable covert communication even in the presence of dishonest authorities.

We introduce the term *ballot freedom* to capture the most useful application of anamorphic voting. Ballot freedom is the ability of a voter to freely communicate their true voting intention to an auditor without the knowledge of the authoritarian regime.

At first glance, ballot freedom seems similar to receipt-freeness or coercion-resistance (by replacing the terms "auditor" with "tallier" and "coercer" with "authoritarian regime"). However, there are at least two key differences. First, receipt-freeness and coercion-resistance are properties that can be enforced by designing a new voting scheme, while ballot freedom is a property achieved through the combination of cryptographic primitives in an existing voting scheme. Second, receipt-free and coercion-resistant voting schemes need to assume the existence of anonymous and/or private channels at some point [9]. This very assumption is ruled out in the context of anamorphic voting, as authoritarian regimes often control all communication channels.

In this paper, we introduce the first constructions for anamorphic voting by investigating combinations of the most common cryptographic primitives in voting schemes. We show that anamorphic voting is possible in the Estonian Internet Voting System (IVXV) [22], CHVote [19], Helios [1] and Belenios [10] voting schemes. Additionally, we demonstrate that different levels of ballot freedom can be achieved depending on the available datasets. To the best of our knowledge, our constructions are the first to demonstrate anamorphism within the context of a combination of cryptographic primitives. Thus, these constructions may also prove useful in domain beyond voting.

2 Anamorphic Voting

We provide a primer of anamorphic encryption, from which we informally introduce anamorphic voting and ballot freedom, leaving a formal treatment for future work. We then discuss how to make CHVote [19] anamorphic, achieving ballot freedom using different cryptographic primitives.

2.1 From Anamorphic Encryption to Anamorphic Voting

Anamorphic encryption [25] enables entities to encrypt hidden messages evading the censorship imposed by a dictator within a well-established public-key cryptosystem. In its stronger formulation, the dictator owns the receiver's secret key (no receiver privacy) and can force the sender to send ciphertexts over a channel controlled by the dictator (no sender freedom), although sender and receiver might have previously exchanged keys over private channels.

In strongly secure anamorphic encryption, the dictator forces the sender to encrypt a message fm using a forced public key fpk for which the dictator knows the corresponding private key fsk. The sender can evade censorship by generating randomness for the encryption using a coin-toss faking algorithm $\mathsf{fRandom}$,

which takes as input fm, fpk, a double public key dpk (with the corresponding double private key dsk known only by the receiver) and a hidden message dm, and generates an anamorphic ciphertext that gives fm when decrypted with fsk, and dm when decrypted with dsk by the receiver.

In anamorphic voting, we extend the concept of anamorphic encryption to support the multiple cryptographic primitives that constitute a voting scheme, that is, we split anamorphic encryptions across multiple components, such as multiple ciphertexts, ciphertext and signature, or ciphertext and proof of knowledge. This approach allows entities to encrypt hidden messages, evading the censorship imposed by dishonest authorities within a well-established voting scheme.

Anamorphic voting is agnostic about the participants. It can apply to scenarios where, for instance, a voter wants to communicate with another voter or an honest authority with another honest authority. To address the specific case of voters conveying their true vote intentions to an auditor despite the presence of dishonest authority, we introduce the concept of ballot freedom.

Ballot freedom assumes the following strong threat model:

- The auditor double public key dpk is known to everyone, including voters and authority.
- The authority executes the voting scheme according to its prescribed specifications, including setup parameters.
- The authority knows at least the election decrypting key and may hold additional keys based on the roles in the voting scheme.
- The authority has access to all data generated within the voting scheme.
- The voter can only communicate through channels controlled by the authority (i.e. no anonymous, private, or untappable channels exist).

A notable distinction from anamorphic encryption is that ballot freedom does not assume the existence of private channels.

2.2 A Running Example: CHVote

CHVote [19] has been one of the two major e-voting system proposals in Switzerland. The system is universally verifiable, which means that independent verifiers can confirm the correctness of the election results. Verification ensures that only votes cast by eligible voters are included, that no voter casts more than one vote, and that every valid vote is tallied as recorded.

The election process in CHVote involves several key parties: Election Administrator, Election Authority, Printing Authority, Voter, and Verifier. CHVote is designed to support various election formats, including multiple elections and multiple answers. For simplicity, we focus on the case where voters cast a vote for a single candidate among n possible candidates. It relies on several cryptographic primitives, including ElGamal encryption [15], Schnorr signatures, and Schnorr identification proofs [26].

CHVote can be divided in three phases as follows:

Preparation Phase. The election administrator defines the election parameters, such as the number of candidates n, and works with the election authorities to generate a public encryption key pk. Each authority prepares partial verification codes for each voting option, which are hashed and combined into a single verification code v_c. The printing authority collectively generates and prints eligibility, confirmation, and validity data on the voter's election card.

Vote Casting Phase. To cast a vote, the voter provides their choice m to the voting client, which generates a ballot $\alpha = (\hat{x}, b, pk)$. Here, $\hat{x}$ is the voter's public credential derived from their private credential x, and $b = (b_1, b_2) = \mathsf{Enc}(pk, m; r) = (g^r, m \cdot pk^r)$ is an Oblivious Transfer (OT) query constructed using ElGamal encryption. The ballot also includes a non-interactive zero knowledge proof (NIZKP). The authorities, acting as OT senders, respond to this query, enabling the voting client to derive verification codes for the chosen candidate. Additionally, the ballot contains a Schnorr-like identification proof linking the public credential $\hat{x}$ and the encrypted vote. Let H be a secure hash function, and let $\hat{g}$ be the generator of $\mathbb{Z}_{\hat{p}}$, where $\hat{q} \mid \hat{p} - 1$, the NIZKP, consisting of three combined proofs, is generated as follows:

- pick $w_1 \xleftarrow{\$} \mathbb{Z}_{\hat{q}}, \quad w_2 \xleftarrow{\$} \mathbb{Z}_p, \quad w_3 \xleftarrow{\$} \mathbb{Z}_q$
- compute $t_1 = \hat{g}^{w_1} \mod \hat{p}, \quad t_2 = w_2 \cdot pk^{w_3} \mod p, \quad t_3 = g^{w_3} \mod p$
- compute $s = H(\alpha, t_1, t_2, t_3)$
- compute $r_{v_1} = w_1 - s \cdot x \mod \hat{q}, \quad r_{v_2} = w_2 \cdot m^{-s}, \quad r_{v_3} = w_3 - s \cdot r \mod q$
- return $\pi = (r_v, s) = ((r_{v_1}, r_{v_2}, r_{v_3}), s)$

To verify the proofs, one computes $t'_1 = \hat{x}^s \cdot \hat{g}^{r_{v_1}}$, $t'_2 = b_2^s \cdot r_{v_2} \cdot pk^{r_{v_3}}$, and $t'_3 = b_1^s \cdot g^{r_{v_3}}$, and then checks if $s = H(\alpha, t'_1, t'_2, t'_3)$.

Tally Phase. After the election, authorities anonymize the ballots via a verifiable mixnet and collectively decrypt them using private key shares. The administrator finalizes the decryption and publishes the results, which independent verifiers validate for correctness and integrity. Several data sets, including encrypted votes and their proofs, are published for auditing purposes, as detailed in Table 7.2 of [19].

2.3 Ballot Freedom in CHVote

CHVote achieves ballot freedom thanks to an anamorphic construction that combines ElGamal and the Schnorr-like identification proof as follows.

Construction 1 (combining ElGamal ans Schnorr). The voter chooses a random $w_1 \xleftarrow{\$} \mathbb{Z}_{\hat{q}}$ and then computes the ElGamal randomness as $r := \hat{H}(dpk^{w_1} \cdot g^{dm})$, where $\hat{H}$ is a hash function that maps group elements into the randomness space, $dpk = \hat{g}^{dsk}$ is the auditor double public key, and dm is the double (short) message or private vote chosen by the voter. Then, the voter computes the NIZKP and the ElGamal ciphertext $ct := \mathsf{Enc}(pk, m; r)$ accordingly to the CHVote protocol specifications outlined above.

Given the public available proof $\pi = ((r_{v_1}, r_{v_2}, r_{v_3}), s)$ and ballot $b = (b_1, b_2) = (g^r, m \cdot pk^r)$, the auditor can retrieve the double message dm using the procedure $\mathsf{dDecS}(dsk, (b, (r_{v_1}, s), \hat{x}))$ as detailed in Algorithm 1. This works because $b_1 = g^t = g^{\hat{H}(t_1^{dsk} \cdot g^{dm})} = g^{\hat{H}(\hat{g}^{w_1 \cdot dsk} \cdot g^{dm})} = g^{\hat{H}(dpk^{w_1} \cdot g^{dm})} = g^r$. Note that the authority knows all keys, including dpk, except dsk. The construction establishes a level of robustness, ensuring that the auditor, and only the auditor, can retrieve and understand if the encrypted message contains or not a double message.

This particular construction requires that the voter private credential x is not known to the authority. If so, given the ballot signature (r_v, s) the authority can compute $w_1 := s \cdot x + r_{v_1}$ and check whether $b_1 = g^{\hat{H}(dpk^{w_1} \cdot g^{dm})}$. To avoid this, the voter can instead link the randomness in the ElGamal encryption to the proof of knowledge of that very randomness, namely, r_{v_3}. The voter chooses a random $w_3 \xleftarrow{\$} \mathbb{Z}_q$ and computes $r = \hat{H}(dpk^{w_3} \cdot g^{dm})$. The proof $\pi = ((r_{v_1}, r_{v_2}, r_{v_3}), s)$ is generated as before, and the auditor retrieves the double message dm using $\mathsf{dDecS}(dsk, (b, (\mathbf{r_{v_3}}, s), \mathbf{b_1}))$.

Algorithm 1 $\mathsf{dDecS}(dsk, (b, (r_{v_1}, s), \hat{x}))$ // (Schnorr)

```
b := {b_1, b_2}
t_1 := x̂^s · ĝ^{r_{v_1}}
for i ∈ M do
    t := Ĥ(t_1^{dsk} · g^i)
    if b_1 = g^t then
        return i
    end if
end for
return ⊥
```

3 More Anamorphic Constructions

We provide a few more generic anamorphic constructions that can be used to achieve ballot freedom in other voting schemes.

Common to all the constructions are the forced message $fm \in \mathbb{M}$, the attacker key pair $(fpk, fsk) \leftarrow \mathsf{EKeyGen}()$, the receiver double key pair $(dpk, dsk) \leftarrow \mathsf{EKeyGen}()$, the double message $dm \in \mathbb{M}$, and a hash function $\hat{H}$ that maps group elements into the randomness space. $\mathsf{EKeyGen}()$ is the ElGamal key generation function that outputs a pair of keys $(sk, pk = g^{sk})$ where $sk \xleftarrow{\$} \mathbb{Z}_p$.

Also here the attacker knows all keys except dsk and all constructions establish robustness, ensuring that decrypting anamorphic ciphertexts with the correct double key results in a valid message, otherwise an explicit abort signal is returned.

Construction 2 (combining two ElGamal ciphertexts). The sender chooses a random $r_1 \xleftarrow{\$} \mathbb{Z}_q$ and computes $ct_1 = \mathsf{Enc}(fpk, fm; r_1)$. Given another forced

Algorithm 2 $\mathsf{dDec}(dsk, (ct_1, ct_2))$ (ElGamal)

1: $ct_1 := (c_{11}, c_{12})$
2: $ct_2 := (c_{21}, c_{22})$
3: $t := c_{11}^{dsk}$
4: **for** $i \in \mathbb{M}$ **do**
5: $\quad k := \hat{H}(t \cdot g^i)$
6: $\quad$ **if** $c_{21} = g^k$ **then**
7: $\quad\quad$ **return** i
8: $\quad$ **end if**
9: **end for**
10: **return** $\perp$

Algorithm 3 $\mathsf{fRandom}(dpk, dm)$ // (ℓ-bit message)

1: $dm := \mathsf{b}_1 \| \mathsf{b}_2 \| \cdots \| \mathsf{b}_\ell \in \{0,1\}^\ell$
2: **for** $i \in [\ell + 1]$ **do**
3: $\quad$ **if** $i = 1$ **then**
4: $\quad\quad r_i \xleftarrow{\$} \mathbb{Z}_q$
5: $\quad$ **else**
6: $\quad\quad r_i := \hat{H}(dpk^{r_{i-1}} \cdot g^{\mathsf{b}_{i-1}})$
7: $\quad$ **end if**
8: **end for**
9: $R \leftarrow \{r_i\}_{i \in [\ell+1]}$
10: **return** R

message $fm' \in \mathbb{M}$, the sender computes $r_2 \leftarrow \hat{H}(g^{dm} \cdot dpk^{r_1})$ and ciphertext $ct_2 = \mathsf{Enc}(fpk, fm'; r_2)$, and sends (ct_1, ct_2). The receiver retrieves dm using the decryption function outlined in Algorithm 2. This works because $c_{21} = g^k = g^{\hat{H}(g^{dm} \cdot c_{11}^{dsk})} = g^{\hat{H}(g^{dm} \cdot dpk^{r_1})} = g^{r_2}$. Note that if $dm \in \mathbb{M} \setminus \{0\}$, r_2 can be computed as $r_2 \leftarrow \hat{H}(dm \cdot dpk^{r_1})$ and the receiver can avoid the exponentiations in step 5 in Algorithm 2.

Since the receiver must compute modular exponentiations to retrieve the message, the double message space is expected to be small, which is generally acceptable in voting, as the message space for the candidates is typically small.

Construction 3 (combining $\ell+1$ ElGamal ciphertexts). Alternatively, the sender can anamorphically send an l-bit long double message dm by casting $\ell+1$ ciphertexts. The sender selects the set of randomness $R := \{r_i\}_{i \in [1,\ell+1]}$ using the fake randomness generator as described in Algorithm 3, which is a simplified version of the sender-anamorphic extension for hybrid PKE with special KEM from [29]. Differently from [29], our fake randomness generator does not require key encapsulation. Algorithm 3 takes in dpk and dm, and outputs R to be used in the ElGamal encryptions. In fact, it creates a link between the randomness used in the encryptions. This is only noticeable by the receiver owning the double secret key dsk. The receiver can decrypt dm using Algorithm 4.

Construction 4 (combining ElGamal and DSA). The last construction consider the DSA signature scheme, which we briefly discuss as follows.

Algorithm 4 $\mathsf{dDecB}(dsk, \mathsf{CT})$ // ℓ-bit message)

$\mathsf{CT} := \{(c_{i1}, c_{i2})\}_{i\in[1,\ell+1]}$
for $i \in [2, \ell+1]$ **do**
  $b_i := \{c_{i1}, c_{i2}\}$
  $t_i^0 := \hat{H}(c_{1i-1}^{dsk})$
  $t_i^1 := \hat{H}(c_{1i-1}^{dsk} \cdot g)$
  if $c_{1i} = g^{t_i^0}$ **then**
    $\mathsf{b}'_{i-1} := 0$
  else if $c_{1i} = g^{t_i^1}$ **then**
    $\mathsf{b}'_{i-1} := 1$
  else
    return $\perp$
  end if
end for
return $\mathsf{b}'_1 \| \mathsf{b}'_2 \| \cdots \| \mathsf{b}'_\ell$

DSA Signature. The DSA signature consists of three algorithms as:

- $\mathsf{DKeyGen}(1^\lambda)$: on input of a security parameter 1^λ, outputs a pair of keys $(vsk, vpk = g^{vsk})$ where $vsk \xleftarrow{\$} \mathbb{Z}_q$, with (p, q, g) satisfying $q \mid (p-1)$ and g being a generator of the subgroup of order q in $\mathbb{Z}_p^*$.
- $\mathsf{DSign}(vsk, m; w)$: which, given a signing key vsk, a message $m \in \mathbb{M}$, and randomness $w \xleftarrow{\$} \mathbb{Z}_q$, and a hash function H, outputs a signature (r_v, s) where $r_v = (g^w \mod p) \mod q$, and $s = w^{-1}(H(m) + vsk \cdot r_v) \mod q$.
- $\mathsf{DVerify}(vpk, m, \sigma = (r_v, s))$: given a verification key vpk, a message m, and a signature σ, outputs $\top$ (valid) if $r, s \in \mathbb{Z}_q \setminus \{0\}$, $v = H(m)s^{-1} \mod q$, $u_1 = g^v \mod p$, $u_2 = vpk^{(r_v \mod q)} \mod p$, and $(u_1 \cdot u_2 \mod p) \mod q = r_v$. Otherwise, it outputs $\perp$.

Algorithm 5 $\mathsf{dDecDSA}(dsk, ct, (r_v, s))$ (DSA)

$ct := \{c_1, c_2\}$
for $i \in \mathbb{M}$ **do**
  $t := \hat{H}(c_1^{dsk} \cdot g^i)$
  if $r_v = g^t$ **then**
    return i
  end if
end for
return $\perp$

The sender chooses a random $r \xleftarrow{\$} \mathbb{Z}_q$ and computes $ct := \mathsf{Enc}(fpk, fm; r)$. Given a (forced) message sm, the sender computes $w := \hat{H}(g^{dm} \cdot dpk^r)$, generates a DSA signature $(r_v, s) := \mathsf{DSign}(vsk, sm; w)$ and sends $(ct, (r_v, s))$. The receiver retrieves dm using the decryption function outlined in Algorithm 5. This works because $r_v = g^t = g^{\hat{H}(g^{dm} \cdot c_1^{dsk})} = g^{\hat{H}(g^{dm} \cdot dpk^r)} = g^w$.

All the constructions above are very efficient. For Construction 1, 2, and 4, the sender (voting device) just needs an extra exponentiation to send an anamorphic ciphertext. All the burden is on the receiver (auditor), who needs to perform a linear number of exponentiations in the size of the double message. Therefore, the double message space for this constructions is expected to be small, like a vote preference. For Construction 3, the sender needs to perform $2 \times \ell$ extra exponentiations per bit-message. For example, to send a 32-bit long double message, the sender needs to perform only 64 extra exponentiations. However, it is different if we consider the burden on the voter rather than the voting device. In this case, it mainly depends on the specificities of the voting system, its implementation, and its user interface.

4 The Estonian Internet Voting System

The Estonian Internet Voting System, IVXV [22], allows voters to cast their vote online. To sign and encrypt a ballot, it uses a digital signature scheme and a homomorphic public-key encryption scheme, which is instantiated with ElGamal. Proof of shuffle and proof of correct decryption are also employed.

4.1 Protocol Description

IVXV is based essentially on the following participants: the election organizer, the registration service, the voter, the vote collector, the ballot box processor (IBBP), the mixing service, and the data auditor. The protocol can be divided into five distinct stages: initialization, voting, post-voting, tallying, and auditing.

Initialization. Before voting begins, the election organizer sets up and configures the IVXV system. In particular, this process includes generating cryptographic keys that will later be used to encrypt and decrypt votes as follows. On input of the security parameter 1^λ, electoral roll $\mathbb{V}$, and candidate list $\mathbb{C}$, the election organizer computes $(pkE_T, skE_T) \xleftarrow{\$} \mathsf{EKeyGen}(1^\lambda)$. Moreover, each voter v has access to their signing key pair spk_v, ssk_v, which were generated by governmental service by computing $(spk_v, ssk_v) \xleftarrow{\$} \mathsf{DKeyGen}(1^\lambda)$. In the IVXV implementation, the signature scheme is RSA-SHA256 (for authentication with ID card or digital ID) or ECDSA-SHA256 (for authentication with mobile ID). Here, we consider only the latter.

Voting. An eligible voter $v \in \mathbb{V}$ encrypts their choice $c_v \in \mathbb{C}$ using the election's public encryption key pkE_T and a randomly generated value r_v resulting in ballot $b = \mathsf{Enc}(pkE_T, c_v, r_v)$. The voter then signs the ballot b using their private signing key ssk_v and a random value rs_v. This creates $vote_v = \mathsf{Sign}(ssk_v, b, rs_v)$. The voter submits their identifier v, a certificate proving their voting eligibility, and $vote_v$ to the vote collector (VC) .

Upon receipt, VC processes the submission and responds with a unique identifier vid, used for (individual) verification of the vote. Note that a voter can cast multiple votes. The set D_{VC} contains all cast votes during this phase, stored without removal.

Post-voting. After the voting phase concludes, IBBP checks the eligibility of each vote and cross-references the registration confirmations, which are stored in D_{RS}, with the data from D_{VC} to ensure consistency. Once this verification is complete, the IBBP compiles a new list of ballots, retaining only the most recent valid vote for each voter and removing those who also cast paper ballots. Then, the IBBP anonymizes the ballots by stripping any voter-identifying information (e.g. signatures) producing B_1. Optionally, the set B_1 can be mixed, producing an output set B_2 along with a proof of correct operation P_{mix} ensuring secure shuffling and re-encryption.

Tallying. The election organizer decrypts each choice c' and computes the voting result. Additionally, it provides a proof of correct decryption P_{dec} along with the plaintext.

Auditing. Different levels of auditing can be performed. Universal verifiability can be achieved by a *complete* audit, in which the auditor has access to all data sets D_{VC}, D_{RS}, B_1, B_2, P_{mix}, P_{dec}. As outlined in [22], a complete audit might leak a lot of information, such as whether a voter re-voted. This increases the risk of coercion, though the dictator already breaks ballot privacy, possibly allowing a complete audit. As an alternative, the auditor can perform a *partial* audit, in which only the data sets B_1, B_2, P_{mix}, P_{dec} are available.

4.2 Ballot Freedom in IVXV

We now explore how ballot freedom can be achieved in the IVXV system across three different scenarios as follows.

Case 1: complete audit with revoting. In this scenario, the auditor has access to all data sets, including D_{VC} containing encrypted votes and their signatures. Revoting allows the voter to cast multiple votes, which the auditor can detect. Ballot freedom can be achieved using the constructions that combine ElGamal ciphertexts outlined in Section 3. According to Construction 2, the voter casts two ballots to communicate a message within a small message space. According to Construction 3, the voter casts $\ell + 1$ ballots to communicate an ℓ-bit-long double message. The auditor can identify all ballots cast by the same voter by examining the signatures.

Case 2: complete audit without revoting. In this scenario, the voter casts only one vote. If the auditor has access to D_{VC}, the voter can link the randomness in the encryption and the randomness in the DSA signature using Construction 4. Interestingly, the IVXV architecture document suggests that the P-384 curve can be used for ElGamal encryption, which is the same curve commonly used in ECDSA. Therefore, they likely share the same parameters, including order and generator. Even if the parameters differ, the hash function in Construction 4 can maps elements between the groups. A practical challenge arises if the signature service is provided by an app that might not allow the voter to choose

the randomness. Therefore, the voter would need to a modified version of the app that supports customized randomness input.

Case 3: partial audit. In this scenario, the auditor has access to anonymized ballots in B_1, therefore the previous approaches cannot straightforwardly signal the voter's real intention. However, if voters reveal their randomness to other voters, they could still signal their true vote intentions by using Construction 2 or Construction 3. The latter construction is more challenging because voters must reveal their randomness in sequence to form a link between them. In both cases, the decryption algorithms dDec and dDecB require the auditor to check all other ballots in B_1 for randomness links. Despite these challenges, ballot freedom remains feasible even in the partial audit scenario.

5 Helios

Helios [1] builds on Benaloh's Simple Verifiable Voting [4], separating ballot preparation and casting. Ballots can be prepared and viewed without authentication, while authentication is required only for casting, ensuring auditability by allowing anyone to verify the system's integrity. In this paper, we focus on Helios 2.0 [2], which, compared to the first protocol, replaces mixnet-based tallying with homomorphic tallying. In addition, voters now provide a NIZKP to demonstrate the validity of their encrypted votes without revealing their content.

The election process in Helios 2.0 is as follows, where the cryptographic descriptions are taken from [12]:

Ballot Casting. The voter selects their choice $m \in \{min, \ldots, max\}$ and forms the vote as $v = g^m$. Then, the voter encrypts v under the trustees' public key pk using ElGamal encryption yielding $ct = (c_1, c_2) = \mathsf{Enc}(pk, v; r) = (g^r, g^m \cdot pk^r)$, and generates a NIZKP showing that ct is a valid encryption for a message m. The proof is generated as follows.

- For all invalid plaintexts $i \in \{m_{min}, \ldots, m_{m-1}, m_{m+1}, \ldots, m_{max}\}$: randomly generate a challenge $s_i \in \mathbb{Z}_q^*$ and a response $r_i \in \mathbb{Z}_q^*$. Compute the witnesses $c_{1i} = g^{r_i}/{c_1}^{s_i}$ and $c_{2i} = pk^{r_i}(c_2/g^i)^{s_i}$.
- For the valid plaintext m: select a random nonce $w \in \mathbb{Z}_q^*$. Compute the witnesses $c_{1m} = g^w$ and $c_{2m} = pk^w$. Derive the challenge s_m using the hash function H:

$$s_m = H(c_{1m_{min}}, c_{2m_{min}}, \ldots, c_{1m_{max}}, c_{2m_{max}}) - \sum_{i \neq m} c_i \mod q.$$

Compute the response $r_m = w + r \cdot s_m$. The proof for the vote is $\pi = (r_m, s_m)$

Before casting, the voter has the option to audit the ballot to confirm it accurately represents their intended vote. During this process, the randomness

used in creating the ballot is revealed, allowing independent verification of its correctness.

Once satisfied, the voter submits their ballot to the election authority. The election authority authenticates the voter, verifies their eligibility, and checks the validity of the NIZKP. A valid ballot is published on the bulletin board along with the voter's identity.

After submission, the voter can verify that their ballot appears on the bulletin board. Observers can confirm that the proofs attached to each ballot are valid, ensuring all published ballots represent legitimate votes.

Tallying. After the voting period ends, the election authority homomorphically combines all the encrypted ballots to produce an encrypted tally. The encrypted tally is published on the bulletin board.

Each trustee publishes a partial decryption of the encrypted tally. Alongside the partial decryption, trustees include a signature of knowledge to prove its correctness. These proofs are publicly available for verification. The election authority decrypts the final tally and publishes the election result.

5.1 Ballot Freedom in Helios

Case 1: audit with revoting. In Helios, the voter can cast multiple ballots, all of which are published on a public bulletin board. Once they authenticate a ballot, that ballot is displayed on the bulletin board as their final choice, while all previously cast ballots are archived. Since the bulletin board data is auditable by anyone at any time, the voter can achieve ballot freedom by using any constructions that combine ElGamal ciphertexts. In Construction 2, the voter casts two ballots, whereas in Construction 3, the voter casts $\ell + 1$ ballots. Note that the voter must be able to freely select the randomness r in the encryption $\mathsf{Enc}(pk, v; r)$.

Case 2: audit without revoting. The voter can also use Construction 1 by using the ElGamal ciphertext and its corresponding proof. The voter selects $w \in \mathbb{Z}_q^*$ and computes the ElGamal randomness as $r = \hat{H}(dpk^r \cdot g^{dm})$. The voter generates the proof $\pi = (r_m, s_m)$, and when the auditor retrieves $ct = (c_1, c_2)$ and its corresponding proof π from the bulletin board, they can decrypt the double message dm using $\mathsf{dDecS}(dsk, (ct, (r_m, s_m)), c_1)$.

6 Belenios

Belenios [10] is an electronic voting system built upon Helios [1] that offers vote privacy and verifiability. Similar to Helios, it uses ElGamal and there is a public bulletin board that contains the accepted ballots and the result of the voting. Schnorr signatures are used to sign an encrypted ballot to avoid ballot stuffing.

6.1 Protocol Description

Due to space limitations, we refer the reader to the original paper [10] for a detailed description of the protocol. Here we provide a brief overview of the protocol that is instrumental to assess ballot freedom. We briefly recall the Schnorr signature scheme as follows.

Schnorr Signature. The Schnorr signature consists of three algorithms as:

- $\mathsf{SKeyGen}(1^\lambda)$: on input of a security parameter 1^λ, outputs a pair of keys $(vsk, vpk = g^{vsk})$ where $vsk \xleftarrow{\$} \mathbb{Z}_q$, with (p, q, g) satisfying $q \mid (p-1)$ and g being a generator of the subgroup of order q in $\mathbb{Z}_p^*$.
- $\mathsf{SSign}(vsk, m; w)$: given a signing key vsk, a message $m \in \mathbb{M}$, and randomness $w \xleftarrow{\$} \mathbb{Z}_q$, and a hash function H, outputs a signature (r_v, s) where $s = H(m|g^w) \mod q$ and $r_v = w - vsk \cdot s \mod q$
- $\mathsf{SVerify}(vpk, m, \sigma = (r_v, s))$: given a verification key vpk, a message m, and a signature σ, outputs $\top$ if $s = H(m|A)$ where $A = vpk^s g^{r_v}$. Otherwise, it outputs $\bot$.
  Note that a variant of the Schnorr signature can be used to prove the knowledge of a discrete logarithm. For example, to prove knowledge of the randomness r used to generate an ElGamal ciphertext $ct = (c_1, c_2) = (g^r, m \cdot pk^r)$, set $vsk = r$ and $vpk = c_1$.

For each voter id, a unique signing key $sk_{id} \in \mathbb{Z}_q$ is generated. Then, the voter V encrypts the vote preference v as $ct = \mathsf{Enc}(pk, v; r)$ and generate a signature $s = \mathsf{SSign}(vsk, ct; w)$, where vsk is their signing key. The voter computes a hash $h = \text{hash}(b)$ to serve as a tracking number. The server appends b to the election data D. At any point, V can verify that h is included in the list of *pretty ballots* (P_B), which consists of the hashes of the final ballots submitted by voters. If the election includes non-homomorphic ciphertexts, the trustees shuffle the ballots. The server homomorphically combines the ciphertexts and sends the result to the trustees to compute the final election outcome.

An auditor can perform verifications during the voting phase and after the tally. During the voting phase, an auditor can retrieve the public board to verify its consistency. They check that for each ballot b, the proofs and signature of b are valid. After the tally, the auditor retrieves the public data D, including the list of ballots (B), shuffles (Σ), partial decryptions (Δ), and the result (res). They verify the consistency of B, ensure it matches previously monitored data, compute $\hat{B} = \text{last}(B)$ (the list of only the latest ballots for each voter), and confirm that the proofs in Σ, Δ, and the result res are valid with respect to $\hat{B}$.

6.2 Ballot Freedom in Belenios

In Belenios, the revoting policy allows voters to cast multiple ballots, with only the last ballot being considered for the tally. If all ballots are publicly available for auditing during the voting phase, a voter can use Construction 2 or Construction 3 as introduced in Section 3 to achieve ballot freedom.

If only the last ballots cast by each voter are available to the auditor, the voter can use Construction 1 to achieve ballot freedom. In doing so, the voter links the randomness in the ElGamal encryption to the Schnorr signature of the ballot. Similar to the case of ballot freedom in CHVote, this construction requires that the voter signing key sk_{id} is unknown to a dishonest authority. If the authority knows the key, given the ballot signature (r_v, s), it can compute $w := s \cdot sk_{id} + r_v$ and check whether the first element of the ElGamal encryption of the vote $ct := (c_1, c_2)$ is $c_1 = g^{\hat{H}(dpk^w \cdot g^{dm})}$. In this case, the voter can instead link the randomness in the ElGamal encryption to the proof of knowledge of that very randomness. The proof is generated as per the Schnorr signature by substituting sk_{id} with the ElGamal randomness r and vpk with c_1.

7 Related Work

Dishonest authorities can compromise the security of voting systems in various ways. Bernhard et al. [7] showed that weak Fiat-Shamir heuristics lead to security breaches in Helios when malicious trustees are present. Similarly, Cortier et al. [11] demonstrated that an attacker who corrupts both the voting server and trustees can break verifiability in Belenios if a variant of Fiat-Shamir heuristic is used. Haines et al. [20] showed that having honest trustees is sometimes insufficient to guarantee ballot privacy. Moreover, ensuring trustworthy trustees is not a trivial task. Benaloh et al. [5] examined the practical challenges that trustees face in preserving the privacy of votes.

There is a long stream of research on covert and subliminal channels in voting [16, 23, 3] but they all focus on the maliciousness of the channel rather than the ability of the voter to communicate their true intention.

Coercion, in general, requires that trust assumptions be carefully vetted. Finogina et al. [17] demonstrated that in coercion-resistant voting, a coercer can utilize new cryptographic tools to prevent voters from evading coercion. Tally-hiding [24, 13] and cleansing-hiding [14] voting schemes typically rely on MPC to mitigate coercion even in the presence of dishonest trustees. All the schemes outlined above assume at least one honest trustee and/or a private or anonymous channel. Budurushi et al. [8] proposed a voting scheme in which the attacker controls all channels but not all tally servers. Similarly, Caveat coercitor [18] does not require a private channel but assumes the existence of at least one honest trustee. Solutions involving DRE machines without tallying servers have been proposed [27, 21]. These schemes assume a private voting booth and that the machines do not reveal the voter's choice. The novel aspect of our work is to demonstrate that a voter can still achieve privacy in communicating with an auditor using an existing voting scheme, even when the attacker controls all communication channels and election keys.

8 Discussion and Future Work

In this work, we introduced the concept of anamorphic voting, which studies whether a combination of cryptographic primitives in a voting scheme can be used to send covert messages in the presence of dishonest authorities. We demonstrated that IVXV, CHVote, Belenios, and Helios each enable voters to freely communicate their true voting intention to an auditor without the knowledge of dishonest authorities, a property we refer to as ballot freedom.

Anamorphic voting is in its infancy, and several future research directions can be explored. One is to investigate ballot freedom in voting schemes recently implemented in real-world elections, such as ElectionGuard [6], or in schemes that rely on cryptographic primitives not covered in this work, such as commitment schemes and lattice-based cryptography. Another direction involves formalizing the concept of ballot freedom to enable proofs of whether a given voting scheme achieves of fails to achieve ballot freedom. Formalization can also help clarify the relationship between ballot freedom and other security properties, such as receipt-freeness. Intuitively, ballot freedom appears to be orthogonal to receipt-freeness: if a voting scheme supports anamorphic voting, a coerced voter could reveal their ballot to the coercer (auditor) as proof! However, our constructions might provide a form of coercion evidence such as in [18] but in a cleaner way.

Our constructions can be viewed as a form of strong anamorphic encryption instantiated across multiple cryptosystems. Exploring this independently of voting is worthwhile, as it may also prove useful for other secure applications.

Acknowledgment. Rakeei and Lenzini's research is supported by the ANR and FNR international project INTER/AN/20/14926102 - "Secure and Veriflable Electronic Testing and Assessment Systems" (SEVERITAS).

References

1. Adida, B.: Helios: Web-based open-audit voting. In: USENIX sec. symp. (2008)
2. Adida, B., De Marneffe, O., Pereira, O., Quisquater, J.J., et al.: Electing a university president using open-audit voting: Analysis of real-world use of helios. EVT/WOTE **9**(10) (2009)
3. Adida, B., Neff, C.: Efficient receipt-free ballot casting resistant to covert channels. USENIX EVT/WOTE (2008)
4. Benaloh, J.: Simple verifiable elections. EVT **6**, 5–5 (2006)
5. Benaloh, J., Naehrig, M., Pereira, O.: Reactive: Rethinking effective approaches concerning trustees in verifiable elections. Cryptology ePrint Archive (2024)
6. Benaloh, J., Naehrig, M., Pereira, O., Wallach, D.S.: ElectionGuard: a cryptographic toolkit to enable verifiable elections. In: USENIX Security) (2024)
7. Bernhard, D., Pereira, O., Warinschi, B.: How not to prove yourself: Pitfalls of the fiat-shamir heuristic and applications to helios. In: ASIACRYPT. Springer (2012)
8. Budurushi, J., Neumann, S., Olembo, M.M., Volkamer, M.: Pretty understandable democracy - a secure and understandable internet voting scheme. In: 2013 International Conference on Availability, Reliability and Security (2013)

9. Chevallier-Mames, B., Fouque, P.A., Pointcheval, D., Stern, J., Traoré, J.: On some incompatible properties of voting schemes. In: Towards Trustworthy Elect. (2010)
10. Cortier, V., Galindo, D., Glondu, S., Izabachène, M.: Election verifiability for helios under weaker trust assumptions. In: ESORICS. pp. 327–344. Springer (2014)
11. Cortier, V., Gaudry, P., Yang, Q.: How to fake zero-knowledge proofs, again. In: E-Vote-Id. Tal Tech Press (2020)
12. Cortier, V., Smyth, B.: Attacking and fixing helios: An analysis of ballot secrecy. Journal of Computer Security **21**(1), 89–148 (2013)
13. Cortier, V., Gaudry, P., Yang, Q.: A Toolbox for Verifiable Tally-Hiding E-Voting Systems, pp. 631–652. ESORICS (2022)
14. Cortier, V., Gaudry, P., Yang, Q.: Is the jcj voting system really coercion-resistant? In: CSF. pp. 186–200 (2024)
15. Elgamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. Information Theory, IEEE Transactions on **31**(4), 469–472 (1985). https://doi.org/10.1109/TIT.1985.1057074
16. Feldman, A.J., Benaloh, J.: On subliminal channels in encrypt-on-cast voting systems. EVT/WOTE, USENIX Association (2009)
17. Finogina, T., Herranz, J., Roenne, P.B.: Expanding the toolbox: coercion and vote-selling at vote-casting revisited. E-Vote-ID (2024)
18. Grewal, G.S., Ryan, M.D., Bursuc, S., Ryan, P.Y.: Caveat coercitor: Coercion-evidence in electronic voting. In: 2013 IEEE Symposium on Security and Privacy
19. Haenni, R., Koenig, R.E., Locher, P., Dubuis, E.: Chvote protocol specification. Cryptology ePrint Archive (2017)
20. Haines, T., Mueller, J., Querejeta-Azurmendi, I.: Scalable coercion-resistant e-voting under weaker trust assumptions. In: SAC (2023)
21. Harrison, L., Bag, S., Hao, F.: Camel: E2e verifiable instant runoff voting without tallying authorities. In: Asia CCS. p. 742–752 (2024)
22. Heiberg, S., Martens, T., Vinkel, P., Willemson, J.: Improving the verifiability of the estonian internet voting scheme. In: E-Vote-ID (2016)
23. Karlof, C., Sastry, N., Wagner, D.: Cryptographic voting protocols: A systems perspective. USENIX Security Symposium (2005)
24. Liedtke, J., Küsters, R., Müller, J., Rausch, D., Vogt, A.: Ordinos: a verifiable tally-hiding electronic voting protocol. In: IEEE EuroS&P (2020)
25. Persiano, G., Phan, D.H., Yung, M.: Anamorphic encryption: private communication against a dictator. In: EUROCRYPT. pp. 34–63. Springer (2022)
26. Schnorr, C.P.: Efficient identification and signatures for smart cards. p. 239–252. CRYPTO (1989)
27. Shahandashti, S., Hao, F.: Dre-ip: A verifiable e-voting scheme without tallying authorities. In: ESORICS. pp. 223–240 (2016)
28. The Economist Group: Democracy Index 2023. https://www.eiu.com/n/campaigns/democracy-index-2023
29. Wang, Y., Chen, R., Huang, X., Yung, M.: Sender-anamorphic encryption reformulated: Achieving robust and generic constructions. In: ASIACRYPT (2023)