SAVER: Snark-friendly, Additively-homomorphic, and Verifiable Encryption and decryption with Rerandomization

Jiwon Lee[1], Jaekyoung Choi[2], Jihye Kim[2], and Hyunok Oh[1]

[1] Hanyang University, Seoul, Korea,
{jiwonlee,hoh}@hanyang.ac.kr
[2] Kookmin University, Seoul, Korea,
{cjk2889,jihyek}@kookmin.ac.kr

Abstract. In the pairing-based zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARK), there often exists a requirement for the proof system to be combined with encryption. As a typical example, a blockchain-based voting system requires the vote to be confidential (using encryption), while verifying voting validity (using zk-SNARKs). In these combined applications, a general solution is to extend the zk-SNARK circuit to include the encryption code. However, complex cryptographic operations in the encryption algorithm increase the circuit size, which leads to impractically large proving time and the CRS size.
In this paper, we propose *Snark-friendly, Additively-homomorphic, and Verifiable Encryption and decryption with Rerandomization* or *SAVER*, which is a novel approach to detach the encryption from the SNARK circuit. The encryption in SAVER holds many useful properties. It is *SNARK-friendly*: the encryption is conjoined with an existing pairing-based SNARK, in a way that the encryptor can prove pre-defined properties while encrypting the message apart from the SNARK. It is *additively-homomorphic*: the ciphertext holds a homomorphic property from the ElGamal-based encryption. It is a *verifiable encryption*: one can verify arbitrary properties of encrypted messages by connecting with the SNARK system. It provides a *verifiable decryption*: anyone without the secret can still verify that the decrypted message is indeed from the given ciphertext. It provides *rerandomization*: the proof and the ciphertext can be rerandomized as independent objects so that even the encryptor (or prover) herself cannot identify the origin.
For the representative application, we define and construct a voting system scenario and explain the necessity of each property in the SAVER. We prove the IND-CPA-security of the encryption, along with the soundness of encryption and decryption proofs. The experimental results show that the voting system designed from our SAVER yields 0.7s proving/encryption (voting) time, and 16MB-sized CRS for the SNARK.

Keywords: pairing-based zk-SNARK, verifiable encryption, verifiable decryption, public-key encryption, additively-homomorphic, rerandomization

1 Introduction

Verifiable encryption (VE) [Ate04, CS03, CD00, LN17, YAS+12] is a cryptographic system where the encrypted data provides a proof that can guarantee publicly-defined properties. It can be a useful primitive in trust-based protocols, such as group signatures or key escrow services. The verifiable property varies depending on the nature of the application. For instance, in the group signature, the verifiable encryption is used for the signer to encrypt and prove its identity commitment, which is evidence for detecting the malicious signer in case of treachery. In the key escrow systems where users deposit their keys to the trusted party, the verifiable encryption can let users prove their legitimacy of encrypted keys to the others.

The zero-knowledge proof (ZKP) system is a primitive where one can prove a knowledge for some pre-defined relation $\mathcal{R}$, without revealing any other information. As in previous definitions [CS03, LN17], the verifiable encryption can be also viewed as an encryption scheme combined with the ZKP system, by considering the encrypted message as an instance which satisfies the pre-defined relation $\mathcal{R}$. In this case, the ZKP relation should be combined with the encryption, settled in advance along with the protocol. For example, in [CS03], the relation is pre-defined as *discrete logarithm problem*; the ciphertext is an encryption of $(m_1, \ldots, m_k)$ such that $\delta = \gamma_1^{m_1} \cdots \gamma_k^{m_k}$ for common inputs $(\delta, \gamma_1, \cdots, \gamma_k)$.

***Universal* VE from zk-SNARKs.** If we consider the ZKP with arbitrary relations, it is possible to construct verifiable encryption with *universal* relations, which can prove *any* desired properties of the message[3]. The ZKP for verifiable encryption requires the following conditions:

1. The ZKP should be *non-interactive*, to be compatible with the ciphertext in non-interactive public-key encryption.
2. The ZKP should guarantee *knowledge-soundness* of the message; it requires at least zero-knowledge arguments of knowledge (zk-AoK).
3. The ZKP should guarantee that the instances for proving the relation are the same as messages in the encryption, i.e., $m = m'$ for $\mathsf{Prove}(m)$ and $\mathsf{Enc}(m')$.

Considering the fact that the proof size determines the ciphertext payload, the most suitable primitive would be zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARK). Specifically, pairing-based zk-SNARKs [PHGR13, Gro16, GM17, BG18, KLO19, Lip19] yields constant-sized proof, regardless of the relation complexities. The pairing-based zk-SNARK can take any pre-defined arithmetic circuit (e.g. quadratic arithmetic program) as an input so that the prover can convince the verifier that the prover indeed evaluates the function

[3] For instance, assume a user who wants to encrypt his identity while proving that his age is over 20. Since most of the existing verifiable encryptions only focus on the *validity* of the ciphertext, they cannot support this type of specific properties. On the other hand, universal verifiable encryption can easily output zero-knowledge proof for the given flexible relation.

correctly. As for the verifiable encryption, if any desired property is included in the zk-SNARK circuit, the proof ensures that the encrypted data satisfies the property in the circuit.

Unfortunately, the naive combination of the zk-SNARK and encryption is beyond practicality, because of the third condition. To satisfy the consistency of m in the third condition, the entire encryption process must be included in the zk-SNARK circuit to ensure that m is an input for both encryption and the relation, which incurs large overhead. This problem has recently been studied in [KZM+15b, KZM+15a], which focused on boosting the performance when including the standard cryptographic protocols in the zk-SNARK circuit. They designed the SNARK-friendly encryption with minimal multiplications since the circuit size in pairing-based zk-SNARKs relies on the number of multiplications. By optimizing the encryption circuit, their experiment result could boost the zk-SNARK with RSA public-key encryption up to the nearly-practical level: 8.9s proving time and 48MB CRS size.

Necessity for an advanced VE. However, in real-world applications, we often need more than simple RSA encryption. The encryption schemes have evolved according to more complex functionality requirements. Even the well-known examples such as key escrow, secret sharing in previous verifiable encryption schemes may require the encryption to be extended to more sophisticated primitives like identity-based encryption (IBE) [BBG05, KLLO18], attribute-based encryption (ABE) [AHL+12], etc., to cover various applications. This might involve some heavy cryptographic operations like pairings or access tree comparisons. In case of adding rerandomization to make encryption unlinkable [PR07], the relation circuit needs to include verification procedure of the proof using the ciphertext as a witness. This requires the zk-SNARK verification to be included in the rerandomization circuit, which becomes impractically heavy due to multiple pairings.

If we build universal verifiable encryption with the general approach of *encryption-in-the-circuit* [KZM+15a, KZM+15b], the efficiency might become unrealistic when the encryption is a bit out of simplicity. For instance, to cover the example of the voting application described in 1.1, the circuit needs to include additively-homomorphic encryption such as Paillier encryption [Pai99], zk-SNARK verification, rerandomization, decryption procedure, etc. All these properties require huge amount of work on the prover's side. In particular, even considering the special elliptic curve group optimized for the zk-SNARK verification [BSCTV17], this still leads to the tremendous increment in proving time and common reference string (CRS) size.

Separating encryption from the circuit. An intriguing idea to deviate from this efficiency problem is to *separate* the encryption from the zk-SNARK circuit. The main purpose of including the encryption in the circuit is to ensure that the same m is used for both $\mathsf{Prove}(\mathsf{m})$ and $\mathsf{Enc}(\mathsf{m}')$ within the relation. If we can prove this consistency with some pre-published commitments, there is no need to include the entire encryption in the circuit anymore. This idea is

well-addressed in Hash&Prove [FFG+16] and Commit&Prove [CFQ19]. In brief, Hash&Prove [FFG+16] tries to detach the hashing of data from the circuit, and prove that the same data is used in both hash and the zk-SNARK circuit. Commit&Prove in LegoSNARK [CFQ19] extends this concept to the general level, which tries to provide a connectivity among separate zk-SNARK circuits. LegoSNARK provides various proof gadgets such as sum-checks and self-permutations so that they can be interconnected with other proof gadgets. However, even LegoSNARK lacks a proof gadget for the encryption; it is a remaining problem to design an efficient Commit&Prove protocol for the connectivity between encryption and zk-SNARKs.

Universal SAVER. We devise a new technique named *SAVER: Snark-friendly, Additively-homomorphic, and Verifiable Encryption and decryption with Rerandomization*, to detach encryption from the zk-SNARK circuit while maintaining the connectivity between them. Intuitively, it can be also viewed as designing an encryption gadget for LegoSNARK's Commit&Prove, where the encryption entails more advanced functionalities. Instead of including the entire complicated encryption in the zk-SNARK circuit, the SAVER provides verifiable encryption conjoined with the existing zk-SNARKs (e.g. [Gro16, GM17, BG18, KLO19]) for a universal relation.

The proposed SAVER is universal verifiable encryption which satisfies zk-SNARK connectivity (SNARK-friendly), additive homomorphism, rerandomizability, and verifiable decryption. We describe each property as follows:

- **SNARK-friendly encryption**: SAVER can be conjoined with zk-SNARK supporting universal relations, which can be realized as universal verifiable encryption. In the encryption, the encryptor can prove any arbitrary predefined relation, while encrypting the message separately from the circuit. Later, the proof and ciphertext are jointly verified to guarantee the relation of the message in the ciphertext.
- **Additively-homomorphic encryption**: an additively - homomorphic encryption is a well-known primitive that allows computations on ciphertexts. SAVER is an additively-homomorphic encryption based on ElGamal encryption variants [CGS97], i.e., $G^{m_1+m_2} = G^{m_1} \cdot G^{m_2}$; the ciphertext can be merged by simple elliptic curve cryptography (ECC) multiplications.
- **Verifiable decryption**: a verifiable decryption [CS03] is a primitive which can convince the verifier that the decrypted message is indeed from the corresponding ciphertext. Likewise, the decryption in SAVER entails a decryption proof, which is verified with message and ciphertext to guarantee the validity. This allows the decryptor to prove the correctness of decrypted messages without revealing her secret key.
- **Rerandomizable encryption**: a rerandomizable encryption [PR07] is a public-key encryption scheme where the ciphertext can be rerandomized, which can be viewed as a newly-encrypted ciphertext. Likewise, ciphertext in the SAVER can be rerandomized as a new unlinkable ciphertext. Since the SAVER outputs an encryption proof as verifiable encryption, the encryption proof is also rerandomized along with the ciphertext.

To justify the practicality, we implemented the proposed SAVER by applying the voting relation in section 1.1. The experiment result yields 0.7s for the voting time, which includes both encryption and zk-SNARK proof. The encryption time takes less than 10ms, which indicates that the additional encryption overhead to the zk-SNARK is almost negligible. The CRS size for the voting relation is only 16MB, and the public key and verification key for the verifiable encryption is from 1MB to 8MB, linearly depending on the message size.

Our contributions. We summarize the contributions of the paper, from various perspectives listed as follows:

- **Universal verifiable encryption:** the proposed Snark-friendly, Additively-homomorphic, and Verifiable Encryption and decryption with Rerandomization (SAVER) is universal verifiable encryption. The SAVER can be connected with zk-SNARKs such as [Gro16] with any universal relation. The ciphertext and the proof guarantee that the message satisfies the pre-defined relation from zk-SNARK.
- **zk-SNARK connectivity:** instead of including the encryption process in the circuit for the universal verifiable encryption, the SAVER detaches encryption from the zk-SNARK circuit with providing connectivity. The verification in SAVER guarantees a linkage between encryption and the relation, as well as knowledge soundness of the proof.
- **Functionalities:** the proposed SAVER supports and satisfies many functionalities. It is *SNARK-friendly*: the encryption is compatible with zk-SNARK composition. It is *Additively-homomorphic*: the ciphertext can be merged additively from the homomorphic property. It is *verifiable encryption*: one can encrypt a message while proving any universal relation for the message. It is *verifiable decryption*: the decryptor can convince the verifier that the decrypted message is indeed from the ciphertext, without revealing her secret key. It provides *rerandomization*: the ciphertext can be rerandomized to be unlinkable to the original one.
- **Vote-SAVER:** to justify the functionalities in SAVER, we define an ideal voting system and propose an efficient Vote-SAVER scheme (in section 1.1). While existing voting systems lack some necessary properties, the Vote-SAVER can efficiently satisfy them all without any compromise.
- **Implementation:** we implement our SAVER with the voting application on the real computer system to show the practicality of the construction. The experiment result yields 0.7s for zk-SNARK proving time and 10ms for encryption, with the CRS size of 16MB for the voting relation.
- **Security:** the proposed SAVER requires many security notions: indistinguishability (IND-CPA), encryption knowledge soundness, rerandomizability, perfect decryption soundness, and perfect zero-knowledge. We formally define each property and provide security proof in a standard model.

The rest of the paper proceeds as follows. Section 1.1 provides a specific application to justify the functionalities in the SAVER. Section 2 organizes related works. In section 3, we describe some necessary preliminaries and definitions for

building blocks. Section 4 represents a formal definition of the proposed SAVER. Section 5 presents the construction of SAVER along with main ideas. In section 7, we present a formal description on the voting application of section 1.1. Section 8 shows experiment results on voting application with implementing our SAVER. In section 9, we draw a conclusion.

1.1 Application: Vote-SAVER

Our proposed SAVER is universal verifiable encryption with many useful functionalities - zk-SNARK connectivity, additive-homomorphism, rerandomizability, and verifiable decryption. To strengthen the justifications on such complex functionalities, we specify one of the interesting applications, *voting*, which is mentioned as a representative example of verifiable encryption in the cryptography encyclopedia [Sak11]. In our observation, existing proposals on voting systems rely on some trusted authority at the end, which cannot fundamentally prevent the *malicious authority* from tampering with the result or at least disdaining the privacy. It turns out that advanced verifiable encryption with the zk-SNARK can resolve this long-lasting problem. We first redefine the essential properties in the voting system, to tackle the holes in various existing systems and elaborate on why they are fundamentally difficult to eliminate. Then we show that the verifiable encryption with the zk-SNARK can provide an efficient solution to satisfy the redefined properties of the fundamentally reliable voting system, where privacy and verifiability can co-exist under any circumstances.

Essential properties. Constructing an ideal voting system, whether offline or online, has been a famous research topic for a long history. It is agreed in common that a reliable voting system must satisfy the following properties, which are well-defined in the surveys [JMP13, AM16]:

- **Vote-integrity:** the entire system should be non-malleable; even the administration must not be able to manipulate the result.
- **Receipt-freeness:** for privacy, it is defined in [AM16] that the voting system should not give the voter any evidence to prove to a third party how she voted. As stated in [AM16], this property implies *ballot-secrecy*.
- **Eligibility verifiability:** an observer should be able to verify that the vote was cast by an eligible voter.
- **Individual verifiability:** a voter should be able to verify that his vote is included in the result.
- **Universal verifiability:** an observer should be able to verify that the result is tallied correctly from the entire votes.

While existing proposals [HS00, Oka97, IKSA03, MN06, Adi08, RBH$^+$09, CRST15, Smy18] capture most of the given properties, they do not fundamentally satisfy all properties at once. For instance, [MN06] relies on the split authorities for privacy, which is breakable when both authorities are corrupted. Helios [Adi08] fails to maintain privacy for dishonest ballots, due to the lack of individual verifiability. In fact, it is recently proved in [CL18] that the privacy

implies individual verifiability, but most of the existing works compromise on the individual verifiability, which leads to some security holes in privacy. It is stated in many works [RS17, Smy18, AM16] that the fundamental achievement of individual verifiability is a difficult open problem, and the most difficult conflict emerges between receipt-freeness and individual verifiability[4]. Even with cryptographic primitives such as ZKPs or blind signatures, it is difficult to prevent the voter (prover) from *reproducing* the proof. Or if a vote is distorted by another entity to prevent reproduction, then it is no longer possible for the voter to identify his vote.

We solve this problem by adopting rerandomizability in verifiable encryption. A rerandomizable encryption is a public-key encryption scheme where the rerandomized ciphertext is independent of the original. By letting the network rerandomize the vote, we can prevent the voter from reproducing the vote because he does not know the new random used in the rerandomization. Nevertheless, he can still *check* the proof to verify that his rerandomized vote is preserved and will be tallied as intended. The remaining properties can also be satisfied, by applying some existing primitives and proposals. Intuitively, we give ideas on how to seize each property:

- **Blockchain (or public bulletin board)** for *vote-integrity:* a blockchain system is well-known for its tamper-proof property; relying on the proof of work (PoW), it is hard to modify the contents in a block once it is fixed. Many systems already adopt the blockchain-based design [LW17], or at least a public bulletin board [Adi08], to ensure the vote-integrity of vote results.
- **Rerandomizable encryption** for *receipt-freeness:* a rerandomizable encryption is a public-key encryption scheme where the ciphertext can be rerandomized, which can be viewed as a newly-encrypted ciphertext. If we allow the blockchain nodes to rerandomize the vote, the voter can no longer reproduce his vote because he does not know the random trapdoor used in the rerandomization.
- **zk-SNARK** for *eligibility verifiability:* the zk-SNARK can be utilized to prove the *membership test*, which is a building block in anonymous blockchain systems such as Zerocash [BCG+14]. The purpose of the membership test is to prove that the prover belongs to the pre-defined group of users, without revealing the actual identity. A well-known algorithm for the membership test is a Merkle hash tree; for the Merkle root of public keys computed in advance, the prover shows that his secret key (corresponding to the public key) generates the same Merkle root value along with its co-paths. By adopting the membership test as a relation, the voter can prove his membership within the public key list without revealing the secret value, while an observer can still verify the eligibility of the voter.

[4] In an abstract point of view, the conflict between privacy and eligibility is rather easy to solve with some cryptographic primitives; many schemes already adopt ZKPs [RBH+09, LW17] or blind signatures [Oka97, IKSA03] to construct a privacy-preserving and verifiable voting system.

– **Verifiable encryption** for *individual verifiability:* the verifiable encryption can resolve the conflict between receipt-freeness and individual verifiability. The main conflict was from the fact that distorting the vote makes the voter difficult to identify his vote. However, if verifiable encryption is combined with the rerandomization, the rerandomized proof ensures that the ciphertext of which message satisfies the relation is correctly rerandomized. This can convince the voter that no manipulations have been done except rerandomization, and his vote will be tallied as he cast.
– **Additively-homomorphic encryption & verifiable decryption** for *universal verifiability:* verifiable decryption can convince the verifier that the decrypted message is indeed from the corresponding ciphertext. The idea is already discussed in [HS00]; if the result can be merged with the additively-homomorphic encryption and the message can be verified with the verifiable decryption, an observer can verify that the decrypted result from the administrator is indeed from the merged ciphertext.

Overall, to satisfy all the given properties, it is required to design a public-key encryption system which satisfies the notion of *rerandomizable encryption*, *verifiable encryption*, *additively-homomorphic encryption*, and *verifiable decryption*, along with the zk-SNARK system. Therefore, we emphasize that *advanced verifiable encryption* with rerandomizability, additive-homomorphism, and verifiable decryption is necessary to construct an ideal voting system.

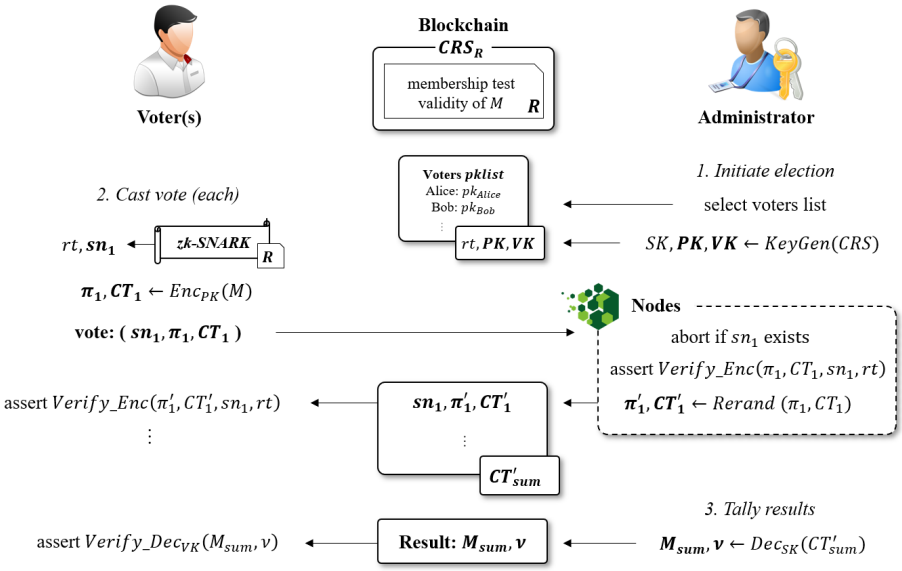

Fig. 1: The Vote-SAVER framework from the advanced verifiable encryption with rerandomizability, additive-homomorphism, and verifiable decryption

Scenario. Figure 1 represents how to efficiently proceed a voting scenario by utilizing the advanced verifiable encryption. The system works with a publicly available blockchain, where the consensus block defines the relation $\mathcal{R}$ of membership test and message validity, with the corresponding common reference string $CRS_{\mathcal{R}}$ generated from zk-SNARK setup. There are two entities, voters and an administrator, who interact mainly through the blockchain subscription. We refer to the election committee as an *administrator*, rather than authority, because the administrator is only responsible for tallying the anonymous results; even when corrupted, she holds no power to trace or manipulate the votes at any cost. Before proceeding the system, we refine the definition of *eligibility* in eligibility verifiability more specifically:

- **Eligibility:** the right to vote must be originated from the voter, not from the authority.

We emphasize that this property is important when applying the voting system to the real world since corrupted authorities often try to tamper the result with impersonation. For instance, some systems [MN06] let the authority pre-define the ballot and distribute them to the users; this cannot prevent the authority from creating *dummies* to inject malicious votes. Therefore, we insist it is also essential to assure that the voter's right should be preserved by himself, not from the authority's setup. Our voting system can satisfy this by letting each user publish his own pk to the public, where pk is generated from user's secret value. For example, a simple way is to let $pk = H(sk)$ for collision-resistant hash H. Without knowing sk, no one can make a valid ballot.

Initiating election. First, to open an election, the administrator makes the $pklist$ of the voters, which prescribes the selection of eligible voters who participate in the election. Then she generates a secret key SK, a public key PK, and a verification key VK for the occasion, to publish PK, VK on the blockchain along with the $pklist$ and its Merkle root rt. This set of PK, VK and $pklist, rt$ defines each election; a new election can be initiated with a different set of PK', VK' and $pklist', rt'$.

Casting votes. After the election is initiated, voters who are selected in the list can cast a vote. Each voter must encrypt the vote and prove the relation (i.e. membership test and message validity) at the same time, via universal verifiable encryption from zk-SNARK. Similar to the membership test in Zerocash [BCG$^+$14], the zk-SNARK circuit outputs a Merkle root rt to prove the belonging within the $pklist$, and a serial number sn to prevent the duplication. Note that the sn does not reveal the identity; it is only used for checking the duplication. As a ballot, a set of serial number sn, proof π and ciphertext $\mathcal{CT}$ is sent to the blockchain network as a transaction. The blockchain node checks if sn already exists in the blockchain (then abort). If sn is unique, it first verifies the proof, rerandomizes the vote from $\pi, \mathcal{CT}$ to $\pi', \mathcal{CT}'$, and publishes (by mining the block) the renewed vote $sn, \pi', \mathcal{CT}'$ on the blockchain. The voter verifies $\pi', \mathcal{CT}'$ for his sn within the verifiable encryption, to be convinced that his vote

is included. This satisfies the individual verifiability, but the voter can only check the existence of his vote; $\pi', \mathcal{CT}'$ is unlinkable from $\pi, \mathcal{CT}$, which also achieves the receipt-freeness.

Tallying results. After all the votes from participants are posted on the blockchain, the administrator closes the vote by declaring the tally result. Since the encryption scheme is additively-homomorphic, anyone can get the merged ciphertext $\mathcal{CT}_{sum}$. The administrator is responsible for decrypting the $\mathcal{CT}_{sum}$ with her own SK, and publishing the corresponding vote result M_{sum} along with the decryption proof ν. By verifying M_{sum}, ν with the verifiable decryption, anyone can be convinced that the result is tallied correctly (universal verifiability).

We define the relation for the voting scenario in section 8, and also provide implementation results of the entire voting system on the real machine.

2 Related Work

We briefly organize related works on two individual topics: zero-knowledge succinct non-interactive argument of knowledge - an essential building block for the SAVER, and reliable voting systems - a suitable application for the SAVER.

zk-SNARKs. A zero-knowledge succinct non-interactive argument of knowledge (zk-SNARK) is introduced in [BCCT12], as a proof system where a prover can generate a proof that they know a witness to an instance in a manner which is *succinct*: proofs are short and verifier computation is small, and *zero-knowledge*: proofs do not reveal the witness. Since Gennaro et al. [GGPR13] introduced a notion of quadratic arithmetic program (QAP), a *pairing-based* zk-SNARKs [Gro16, GM17, BG18, Lip19, KLO19] have received significant attention for their constant sized proof and verification. Groth's protocol [Gro16] set an efficient standard, by yielding three group elements as a proof. Then Groth and Maller [GM17] introduced a notion of simulation-extractability, to prevent malleability in the proof of [Gro16]. However, to achieve simulation-extractability, [GM17] requires a square arithmetic program (SAP) instead of QAP, which doubles the circuit size - which sacrifices proving time and CRS size. To address this issue, Bowe and Gabizon [BG18] applied random oracle to [Gro16], which can transform the [Gro16] to be simulation-extractable. However, this compromises the proof size to five elements. Lipmaa [Lip19] proposed a QAP-based simulation-extractable zk-SNARK with four elements, from the help of more general assumption. Recently, Kim et al. [KLO19] devised the most efficient simulation-extractable zk-SNARK, which achieves both QAP and three elements as a proof, compatible to non simulation-extractable [Gro16].

Voting Systems. Designing an ideal voting system which satisfies both verifiability and privacy was a long-lasting challenge. Since Benaloh and Tuinstra [BT94] introduced a concept of receipt-freeness, it has been agreed as an essential property which implies the voter's privacy. In the past, it was under some strong physical assumption, such as existence of a voting booth [Oka97]. Since then, numerous works [CGS97, HS00, Oka97, IKSA03, MN06, Adi08, RBH$^+$09,

CRST15, Smy18] focused on designing a receipt-free voting systems, by applying variant of cryptographic primitives. Cramer et al. [CGS97] and Hirt et al. [HS00] adopted an ElGamal-based additively-homomorphic encryption for the anonymous tallying. Okamoto [Oka97] and Ibrahim et al. [IKSA03] applied blind signatures, for the eligibility checks of voters. Moran and Naor [MN06] utilizes a permutation on the physical ballot paper for privacy. Later, Helios [Adi08] and vVote [CRST15] received attentions for the practical implementation. Recently, Liu and Wang [LW17] proposed an efficient e-voting protocol based on the blockchain system, which is now being implemented on the Ethereum.

3 Preliminaries

3.1 Notations

In this section, we define some essential notations. For the simple legibility, we define $y_i(x) = \frac{\beta u_i(x)+\alpha v_i(x)+w_i(x)}{\gamma}$ as in [BG18], where $\frac{\beta u_i(x)+\alpha v_i(x)+w_i(x)}{\gamma}$ is from pairing-based SNARK relations such as [Gro16]. Then, for the simplicity, we define $G^{\frac{\beta u_i(x)+\alpha v_i(x)+w_i(x)}{\gamma}}$ in [Gro16] as $G_i = G^{y_i(x)}$.

We use $\{x_i\}$ for the list of elements, which is equivalent to a vector. We also define $[\![X]\!] = span\{X\}$ as a linear combination of $x \in X$, i.e., $[\![X]\!] = \{\sum_{x_i \in X} \eta_i x_i\}$. For any set $[\![X]\!]$, we define $[\![A]\!] \times [\![B]\!] = \{a \cdot b \mid a \in [\![A]\!],\ b \in [\![B]\!]\}$ and $[\![A]\!]^{-1} = \{a^{-1} \mid a \in [\![A]\!]\}$. For any given vectors, $\circ$ represents a Hadamard product (i.e. let $\overrightarrow{a} = (a_1, a_2)$ and $\overrightarrow{b} = (b_1, b_2)$, then $\overrightarrow{a} \circ \overrightarrow{b} = (a_1 \cdot b_1, a_2 \cdot b_2)$).

3.2 Relations

Given a security parameter 1^λ, a relation generator $\mathcal{RG}$ returns a polynomial time decidable relation $\mathcal{R} \leftarrow \mathcal{RG}(1^\lambda)$. For $(\Phi, w) \in \mathcal{R}$ we say w is a witness to the statement (I/O) Φ being in the relation. The statement Φ in the SAVER consists of $\Phi = M \cup \hat{\Phi}$ for message statements $M = \{m_1, \ldots, m_n\}$ arbitrary statements $\hat{\Phi} = \{\phi_{n+1}, \cdots, \phi_l\}$, where l is the number of statements.

3.3 Bilinear Groups

Definition 1. *A bilinear group generator $\mathcal{BG}$ takes a security parameter as input in unary and returns a bilinear group $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, aux)$ consisting of cyclic groups $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ of prime order p and a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ possibly together with some auxiliary information (aux) such that:*

- *there are efficient algorithms for computing group operations, evaluating the bilinear map, deciding membership of the groups, and for sampling the generators of the groups;*
- *the map is bilinear, i.e., for all $G \in \mathbb{G}_1$ and $H \in \mathbb{G}_2$ and for all $a, b \in \mathbb{Z}$ we have*

$$e(G^a, H^b) = e(G, H)^{ab};$$

– *and the map is non-degenerate (i.e., if* $e(G, H) = 1$ *then* $G = 1$ *or* $H = 1$*).*

Usually bilinear groups are constructed from elliptic curves equipped with a pairing, which can be tweaked to yield a non-degenerate bilinear map. There are many ways to set up bilinear groups, both as symmetric bilinear groups, where $\mathbb{G}_1 = \mathbb{G}_2$, and as asymmetric bilinear groups, where $\mathbb{G}_1 \neq \mathbb{G}_2$. We will be working in the asymmetric setting, in what Galbraith, Paterson, and Smart [GPS08] call the Type III setting where there is no efficiently computable non-trivial homomorphism in either direction between $\mathbb{G}_1$ and $\mathbb{G}_2$. Type III bilinear groups are the most efficient type of bilinear groups and hence the most relevant for practical applications.

3.4 Complexity Assumptions

We use *Power Knowledge of Exponent (d-PKE) with Batch Knowledge Check* assumption [Gab19]. In [Gab19] (lemma 2.3), it is proven that the *d-PKE* can be used to *batch* knowledge checks, stated as below:

Lemma 1. batch-PKE *[Gab19]: Assuming the d-PKE the following holds. Fix* $k = poly(\lambda)$*, a constant* t *and an efficiently computable degree* d *rational map* $S : \mathbb{F}^{t+1} \to \mathbb{F}^M$*. Fix any* $i \in [k]$*. For any efficient* $\mathcal{A}$ *there exists an efficient* $\chi_{\mathcal{A}}$ *such that the following holds. Consider the following experiment.* $\alpha_1, \ldots, \alpha_k, \tau \in \mathbb{F}$ *and* $\boldsymbol{x} \in \mathbb{F}^t$ *are chosen uniformly.* $\mathcal{A}$ *is given as input* $[S(\tau, \boldsymbol{x})]$ *and* $\{[\alpha_j \cdot \tau^l]\}_{j\in[k],l\in[0..d]}$ *and outputs a sequence of elements* $([a_1], \ldots, [a_k], [b])$ *in* $\mathbb{G}$*.* $\chi_{\mathcal{A}}$*, given the same input as* $\mathcal{A}$ *together with the randomness of* $\mathcal{A}$ *and* $\{\alpha_j\}_{j\in[k]\setminus\{i\}}$*, outputs* $\mathcal{A}(X) \in \mathbb{F}[X]$ *of degree at most* d *such that the probability that both*

1. $\mathcal{A}$ *"succeeded", i.e.,* $b = \sum_{j=1}^{k} \alpha_j \cdot a_j$*. But,*
2. $\chi_{\mathcal{A}}$ *"failed", i.e.,* $a_i \neq [\mathcal{A}(\tau)]$*.*

is $\mathbf{Adv}^{\text{batch-PKE}}_{\mathcal{R},\mathcal{A},\chi_{\mathcal{A}}}(\lambda) = negl(\lambda)$.

We also introduce a decisional version of the polynomial (Poly) assumption, which is originated from the computational Poly assumption adopted in [GM17]. In the univariate case, the Poly assumption states that for any $G \in \mathbb{G}_1$, given $G^{g_1(\boldsymbol{x})}, \ldots, G^{g_I(\boldsymbol{x})}$, an adversary cannot compute $G^{g_c(\boldsymbol{x})}$ for a polynomial g_c that is linearly independent from $g_1, \ldots, g_I$ - even if it knows $H^{g_c(\boldsymbol{x})}$ for $H \in \mathbb{G}_2$.

We extend the computational Poly assumption to the decisional Poly assumption (D-Poly). In the D-Poly game, the adversary acts similarly as in computational Poly game, except that it queries a challenge polynomial and guesses the nature of the output (i.e. whether the output is generated from the polynomial or from an independent random). In this case, the restriction for the challenge $g_c \notin [\![Q_1]\!]$ is not sufficient where $Q_1 = \{g_1, \ldots, g_I\}$. For example, the adversary should not have $H^{g_c(\boldsymbol{x})}$; otherwise it can check whether the received challenge T is $G^{g_c(\boldsymbol{x})}$ or a random group element by applying pairings (i.e. check the nature of T by $e(T, H^{g_c(\boldsymbol{x})})$)[5]. Thus, the restriction should be extended to

[5] This problem is similar to the decisional BDH assumption: it cannot follow the standard DDH as $(g^a, g^b, T_0 \leftarrow g^z, T_1 \leftarrow g^{ab}, b \leftarrow \{0,1\} \mid b' \leftarrow \mathcal{A}(g^a, g^b, T))$, because the adversary can test if $e(g^a, g^b) \stackrel{?}{=} e(g, T)$.

$H \in \mathbb{G}_2$, to prevent the adversary from obtaining the span of $g_c(\boldsymbol{x})$ in $\mathbb{G}_2$. The formal description of the $D-Poly$ is as follows.

Assumption 1. *Let $\mathcal{A}$ be a PPT adversary, and define the advantage $\mathbf{Adv}^{D-Poly}_{\mathcal{BG},d(\lambda),q(\lambda),\mathcal{A}}(\lambda) = Pr[\mathcal{G}^{D-Poly}_{\mathcal{BG},d(\lambda),q(\lambda),\mathcal{A}}] - \frac{1}{2}$ where $\mathcal{G}^{D-Poly}_{\mathcal{BG},d(\lambda),q(\lambda),\mathcal{A}}$ is defined as below and Q_1, Q_2 is the set of polynomials $g_i(X_1,\ldots,X_q), h_i(X_1,\ldots,X_q)$ queried to $\mathcal{O}^1_{G,\boldsymbol{x}}, \mathcal{O}^2_{H,\boldsymbol{x}}$.*

$\underline{\mathsf{MAIN}\ \mathcal{G}^{D-Poly}_{\mathcal{BG},d(\lambda),q(\lambda),\mathcal{A}}(\lambda)}$
$(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, aux) \leftarrow \mathcal{BG}(1^\lambda);$
$G \leftarrow \mathbb{G}_1; H \leftarrow \mathbb{G}_2; \boldsymbol{x} \leftarrow (\mathbb{Z}^*_p)^q$
$g_c(X_1,\ldots,X_q) \leftarrow \mathcal{A}^{\mathcal{O}^1_{G,\boldsymbol{x}},\mathcal{O}^2_{H,\boldsymbol{x}}}(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, aux)$
where $g_c(\boldsymbol{x}) \notin [\![Q_1]\!] \times [\![Q_2]\!] \times [\![Q_2]\!]^{-1}$
set $T_1 \leftarrow G^{g_c(\boldsymbol{x})}, T_0 \xleftarrow{\$} \mathbb{G}_1$
$b \leftarrow \{0,1\}, T = T_b$
$b' \leftarrow \mathcal{A}^{\mathcal{O}^1_{G,\boldsymbol{x}},\mathcal{O}^2_{H,\boldsymbol{x}}}(T)$
return 1 *if* $b = b'$
else return 0

$\underline{\mathcal{O}^1_{G,\boldsymbol{x}}(g_i)}$
assert $g_i \in \mathbb{Z}^*_p[X_1,\ldots,X_q]$
assert $deg(g_i) \le d$
return $G^{g_i(\boldsymbol{x})}$

$\underline{\mathcal{O}^2_{H,\boldsymbol{x}}(h_j)}$
assert $h_j \in \mathbb{Z}^*_p[X_1,\ldots,X_q]$
assert $deg(h_j) \le d$
return $H^{h_j(\boldsymbol{x})}$

The $(d(\lambda), q(\lambda)) - D - Poly$ assumption holds relative to $\mathcal{BG}$ if for all PPT adversaries $\mathcal{A}$, we have $\mathbf{Adv}^{D-Poly}_{\mathcal{BG},d(\lambda),q(\lambda),\mathcal{A}}(\lambda)$ is negligible in λ.

3.5 Zero-Knowledge Succinct Non-interactive Arguments of Knowledge

For the paring-based zk-SNARK, we adopt the definitions from [Gro16, GM17].

Definition 2. *A zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARK) for $\mathcal{R}$ is a set of four algorithms $\Pi_{\mathsf{snark}} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Vfy}, \mathsf{SimProve})$ working as follows:*

- $(CRS, \tau) \leftarrow \mathsf{Setup}(\mathcal{R})$*: takes a relation $\mathcal{R} \leftarrow \mathcal{RG}(1^\lambda)$ as input and returns a common reference string CRS and a simulation trapdoor τ.*
- $\pi \leftarrow \mathsf{Prove}(CRS, \Phi, w)$*: takes a common reference string CRS, a relation $\mathcal{R}$, a statement and witness in the relation $(\Phi, w) \in \mathcal{R}$ as inputs, and returns a proof π.*

- $0/1 \leftarrow \mathsf{Vfy}(CRS, \Phi, \pi)$*: takes a common reference string* CRS*, a statement* Φ*, a proof* π *as inputs and returns 0 (reject) or 1 (accept).*
- $\pi \leftarrow \mathsf{SimProve}(CRS, \tau, \Phi)$*: takes a common reference string* CRS*, a simulation trapdoor* τ*, a statement* Φ *as inputs and returns a proof* π*.*

It satisfies completeness, knowledge soundness, zero-knowledge, and succinctness described as below:

Completeness: Given a true statement, a prover with a witness can convince the verifier. For all $\lambda \in \mathbb{N}$, for all $\mathcal{R}$ and for all $(\Phi, w) \in \mathcal{R}$, $Pr[(CRS, \tau) \leftarrow \mathsf{Setup}(\mathcal{R}), \pi \leftarrow \mathsf{Prove}(CRS, \Phi, w) : \mathsf{Vfy}(CRS, \Phi, \pi) = 1] = 1$.

Computational Knowledge Soundness: Computational knowledge soundness says that the prover must know a witness and such knowledge can be efficiently extracted from the prover by a knowledge extractor. Proof of knowledge requires that for every adversarial prover $\mathcal{A}$ generating an accepting proof, there must be an extractor $\chi_{\mathcal{A}}$ that, given the same input of $\mathcal{A}$, outputs a valid witness. Formally, an argument system Π_{snark} is computationally considered as knowledge sound if for any PPT adversary $\mathcal{A}$, there exists a PPT extractor $\chi_{\mathcal{A}}$, such that $\mathbf{Adv}^{sound}_{\Pi_{\mathsf{snark}},\mathcal{A},\chi_{\mathcal{A}}}(\lambda)$ is negligible.

$$\mathbf{Adv}^{sound}_{\Pi_{\mathsf{snark}},\mathcal{A},\chi_{\mathcal{A}}}(\lambda) = Pr[(CRS, \tau) \leftarrow \mathsf{Setup}(\mathcal{R}), (\Phi^*, \pi^*) \leftarrow \mathcal{A}(CRS), w \leftarrow \chi_{\mathcal{A}}(trans_{\mathcal{A}}) : Vfy(CRS, \Phi^*, \pi^*) = 1 \wedge (\Phi^*, w) \notin \mathcal{R}] = negl(\lambda).$$

Perfect Zero-Knowledge: Perfect zero-knowledge states that the system does not leak any information besides the truth of the statement. This is modelled by a simulator that does not know the witness but has some trapdoor information that enables it to simulate proofs.

Succinctness: Succinctness states that the argument generates the proof of polynomial size in the security parameter, and the verifier's computation time is polynomial in the security parameter and in statement size.

3.6 Additively-Homomorphic Encryption

We adopt the definition of additively-homomorphic encryption from homomorphic ElGamal encryption [CGS97].

Definition 3. *An encryption system* Π_{AH} *is an additively-homomorphic encryption, if it satisfies* **Completeness** *described as follows:*

$$\mathsf{Enc}(M_i) \circ \mathsf{Enc}(M_j) = \mathsf{Enc}(M_i + M_j)$$
$$\mathsf{Dec}(\mathcal{CT}_i) + \mathsf{Dec}(\mathcal{CT}_j) = \mathsf{Dec}(\mathcal{CT}_i \circ \mathcal{CT}_j)$$

for any messages M_i, M_j *and any ciphertexts* $\mathcal{CT}_i, \mathcal{CT}_j$*.*

3.7 Verifiable Encryption

We refine the definition of verifiable encryption by combining the previous definitions in [CD00, LN17]. We mostly follow the definitions in [CD00], but separate the verification phase individual from decryption as in [LN17].

Definition 4. *A public-key encryption scheme Π_{VE} is a secure verifiable encryption, if it includes the following polynomial-time algorithm for some pre-defined relation $\mathcal{R}$:*

- $\pi, \mathcal{CT} \leftarrow \mathsf{Enc}(PK, M)$: *the encryption of a message M under the public key PK must output a proof π, along with the corresponding ciphertext $\mathcal{CT}$.*
- $0/1 \leftarrow \mathsf{Verify\_Enc}(VK, \pi, \mathcal{CT})$: *takes a verification key VK, an encryption proof π, a corresponding ciphertext $\mathcal{CT}$ as inputs, and outputs 1 if $\pi, \mathcal{CT}$ is within the relation $\mathcal{R}$, or 0 otherwise.*

which satisfies completeness, encryption soundness, and perfect zero-knowledge as described below:

Completeness: A proof π and a ciphertext $\mathcal{CT}$ must pass the verification if they are honestly generated from a message M which satisfies $M \in \mathcal{R}$, formally as $Pr[(\pi, \mathcal{CT}) \leftarrow \mathsf{Enc}(PK, M), M \in \mathcal{R} : \mathsf{Verify\_Enc}(VK, \pi, \mathcal{CT}) = 1] = 1$.

Encryption Soundness: The advantage of an adversary forging verifying $\pi^*, \mathcal{CT}^*$ where $M \notin \mathcal{R}$ is negligible.

$$\begin{aligned}\mathbf{Adv}^{sound}_{\Pi_{\mathsf{VE}}, \mathcal{A}}(\lambda) = Pr[(SK, PK, VK) \leftarrow KeyGen(\lambda), (\mathcal{CT}^*, \pi^*) \leftarrow \mathcal{A}(PK, VK) : \\ \mathsf{Verify\_Enc}(VK, \pi^*, \mathcal{CT}^*) = 1 \wedge \mathsf{Dec}(SK, \mathcal{CT}^*) \notin \mathcal{R}] = negl(\lambda).\end{aligned}$$

Indistinguishability: Assuming within a same relation $\mathcal{R}$, a verifiable encryption should satisfy IND-CPA of the original public-key encryption, with providing additional information π to the adversary.

3.8 Verifiable Decryption

We refine the definition of verifiable decryption from [CS03]; the definition in [CS03] represents the proof system and the encryption system separately, but we intend to combine them as an encryption scheme with verifying phase. Plus, we strengthen the security notion from decryption soundness to perfect decryption soundness, and introduce a new security notion - perfect zero-knowledge.

Definition 5. *A public-key encryption scheme Π_{VD} is a secure verifiable decryption, if it includes the following polynomial-time algorithm:*

- $M, \nu \leftarrow \mathsf{Dec}(SK, \mathcal{CT})$: *the decryption of a ciphertext $\mathcal{CT}$ outputs a message M, along with the corresponding decryption proof ν.*

– $0/1 \leftarrow \mathsf{Verify\_Dec}(VK, M, \nu, \mathcal{CT})$: *takes a verification key* VK*, a message* M*, a decryption proof* ν*, a ciphertext* $\mathcal{CT}$ *as inputs, and outputs 1 if* M, ν *is a valid decryption for* $\mathcal{CT}$ *or 0 otherwise.*

which satisfies completeness, and perfect decryption soundness, and indistinguishability as described below:

Completeness: A message M and a decryption proof ν must pass the verification, if decrypting $\mathcal{CT}$ with SK outputs M, formally as $Pr[(M, \nu) \leftarrow \mathsf{Dec}(SK, \mathcal{CT}), \mathcal{CT} = \mathsf{Enc}(PK, M) : \mathsf{Verify\_Dec}(VK, M, \nu, \mathcal{CT}) = 1] = 1$.

Perfect Decryption Soundness: The advantage of an adversary forging verifying $M^*, \nu^*, \mathcal{CT}^*$ where M^* is not a decryption of $\mathcal{CT}$ is 0.

$$\mathbf{Adv}^{sound}_{\Pi_{\mathsf{VD}},\mathcal{A}}(\lambda) = Pr[(M^*, \nu^*, \mathcal{CT}^*) \leftarrow \mathcal{A}(SK, PK, VK) : \mathsf{Verify\_Dec}(VK, M^*, \nu^*, \mathcal{CT}^*) = 1 \wedge \mathsf{Dec}(SK, \mathcal{CT}^*) \neq M^*] = 0.$$

Indistinguishability: A verifiable decryption should satisfy IND-CPA of the original public-key encryption, with providing additional information ν to an adversary $\mathcal{A}$, for $\mathcal{A}$'s chosen messages.

3.9 Rerandomizable Encryption

We adopt the definition of rerandomizable encryption from [PR07].

Definition 6. *A public-key encryption scheme* Π_{RR} *is rerandomizable, if it includes the following polynomial-time algorithm:*

– $\mathcal{CT}' \leftarrow \mathsf{Rerandomize}(PK, \mathcal{CT})$: *a randomized algorithm which takes a public key* PK *and a ciphertext* $\mathcal{CT}$ *and outputs another ciphertext* $\mathcal{CT}'$*.*

which satisfies completeness and rerandomizability described as below:

Completeness: For every ciphertext $\mathcal{CT}$ and every $\mathcal{CT}'$ in the support of $\mathsf{Rerandomize}(PK, \mathcal{CT})$, we must have $\mathsf{Dec}(SK, \mathcal{CT}') = \mathsf{Dec}(SK, \mathcal{CT})$.

Rerandomizability: For every plaintext M and every ciphertext $\mathcal{CT}$ in the support of $\mathsf{Enc}(PK, M)$, the distribution of $\mathsf{Rerandomize}(PK, \mathcal{CT})$ is identical to another round of $\mathsf{Enc}(PK, M)$.

4 Definition

We represent the definition of our SAVER: Snark-friendly, Additively-homomorphic, and Verifiable Encryption and decryption with Rerandomization - which satisfies the properties of zk-SNARK Π_{snark}, additively-homomorphic encryption Π_{AH}, verifiable encryption Π_{VE}, verifiable decryption Π_{VD} and rerandomizable encryption Π_{RR} altogether.

Definition 7. *For any arbitrary zk-SNARK relation $\mathcal{R}$ (also noted as* relation*), the* SAVER *consists of seven polynomial-time algorithms as follows:*

- $CRS \leftarrow \mathsf{Setup}(\mathsf{relation})$: *takes an arbitrary relation $\mathcal{R}$ as an input, and outputs the corresponding common reference string CRS.*
- $SK, PK, VK \leftarrow \mathsf{KeyGen}(CRS)$: *takes a CRS as an input, and outputs the corresponding secret key SK, public key PK, verification key VK.*
- $\pi, \mathcal{CT} \leftarrow \mathsf{Enc}(CRS, PK, M, \hat{\Phi}; w)$: *takes CRS, a public key PK, a message $M = m_1, \ldots, m_n$, a zk-SNARK statement $\hat{\Phi} = \{\phi_{n+1}, \ldots, \phi_l\}$, and a witness w as inputs, and outputs a proof π and a ciphertext $\mathcal{CT} = (c_0, \cdots, c_n, \psi)$.*
- $\pi', \mathcal{CT}' \leftarrow \mathsf{Rerandomize}(PK, \pi, \mathcal{CT})$: *takes a public key PK, a proof π, a ciphertext $\mathcal{CT}$ as inputs, and outputs a new proof π' and a new ciphertext $\mathcal{CT}'$ with fresh randomness.*
- $0/1 \leftarrow \mathsf{Verify\_Enc}(CRS, \pi, \mathcal{CT}, \hat{\Phi})$: *takes CRS, a proof π, a ciphertext $\mathcal{CT}$, and a statement $\hat{\Phi} = \{\phi_{n+1}, \ldots, \phi_l\}$ as inputs, and outputs 1 if $\mathcal{CT}, \hat{\Phi}$ is in the relation $\mathcal{R}$, or 0 otherwise.*
- $M, \nu \leftarrow \mathsf{Dec}(CRS, SK, VK, \mathcal{CT})$: *takes CRS, a secret key SK, a verification key VK, and a ciphertext $\mathcal{CT} = (c_0, \cdots, c_n, \psi)$ as inputs, and outputs a plaintext $M = m_1, \ldots, m_n$ and a decryption proof ν.*
- $0/1 \leftarrow \mathsf{Verify\_Dec}(CRS, VK, M, \nu, \mathcal{CT})$: *takes CRS, a verification key VK, a message M, a decryption proof ν, and a ciphertext $\mathcal{CT}$ as inputs, and outputs 1 if M is a valid decryption of $\mathcal{CT}$, or 0 otherwise.*

It satisfies completeness, indistinguishability, encryption knowledge soundness, rerandomizability, decryption soundness, perfect zero-knowledge as described below:

Completeness: The completeness of SAVER must satisfy the completeness of Π_{snark}, Π_{AH}, Π_{VE}, Π_{VD} and Π_{RR} altogether.

Indistinguishability: The indistinguishability is also known as semantic security (IND-CPA). The IND-CPA of the SAVER should be indistinguishability of Π_{VE} and Π_{VD}, which is defined by an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ via following game.

Setup: The challenger $\mathcal{C}$ runs $\mathsf{Setup}(\mathsf{relation})$ to obtain CRS, and share CRS and statements $\hat{\Phi}$ to $\mathcal{A}$.

KeyGen: $\mathcal{C}$ runs $\mathsf{KeyGen}(CRS)$ to obtain a secret key SK, a public key PK, and a verification key VK. Then, $\mathcal{C}$ gives PK,VK to $\mathcal{A}$.

Enc1: For the polynomial-time, $\mathcal{A}$ may issue an encryption query M_i, to obtain the corresponding ciphertext $\mathcal{CT}_i$ and a decryption proof ν_i. As a standard IND-CPA game, $\mathcal{A}$ can encrypt the message by himself with the PK. The purpose of the encryption query is to provide $\mathcal{A}$ an additional information: a decryption proof ν_i. For $\mathcal{A}$'s query M_i, $\mathcal{C}$ generates π_i,$\mathcal{CT}_i$ by running $\mathsf{Enc}(CRS, PK, M_i, \hat{\Phi}; w)$, generates ν_i by running $\mathsf{Dec}(CRS, SK, VK, \mathcal{CT}_i)$, and returns $(\pi_i, \mathcal{CT}_i, \nu_i)$ to $\mathcal{A}$.

Challenge: For the challenge, $\mathcal{A}$ outputs two messages M_0 and M_1. $\mathcal{B}$ picks $b \in \{0,1\}$ to choose M_b, generates $\pi, \mathcal{CT}$ by running $\mathsf{Enc}(CRS, PK, M_b, \hat{\Phi}; w)$, and returns $\pi, \mathcal{CT}$ to $\mathcal{A}$.

Enc2: $\mathcal{A}$ can continue to issue encryption queries M_j, same as Enc1. The only restriction is that $M_j \notin \{M_0, M_1\}$.

Guess: $\mathcal{A}$ outputs its guess $b' \in \{0,1\}$ for b, and wins the game if $b = b'$.

Let $\mathbf{Adv}^{ind}_{\mathsf{SAVER},\mathcal{A}}(\lambda)$ be the advantage of $\mathcal{A}$ winning the above game. For a negligible function ϵ, it is IND-CPA secure if for any adversary $\mathcal{A}$ we have that $|\mathbf{Adv}^{ind}_{\mathsf{SAVER},\mathcal{A}}(\lambda) - 1/2| < \epsilon$.

Encryption Knowledge Soundness: The encryption knowledge soundness is a combined definition of computational knowledge soundness in Π_{snark} and encryption soundness in Π_{VE}. It is formally defined as follows:

$$\begin{aligned}\mathbf{Adv}^{sound}_{\mathsf{SAVER},\mathcal{A},\chi_{\mathcal{A}}}(\lambda) = Pr[&(CRS,\tau) \leftarrow \mathsf{Setup}(\mathcal{R}), (PK, SK, VK) \leftarrow \mathsf{KeyGen}(CRS),\\ &(\pi^*, \mathcal{CT}^*, \hat{\Phi}^*) \leftarrow \mathcal{A}(CRS, PK, VK), (M, w) \leftarrow \chi_{\mathcal{A}}(trans_{\mathcal{A}}) :\\ &\mathsf{Verify\_Enc}(CRS, \pi^*, \mathcal{CT}^*, \hat{\Phi}^*) = 1 \wedge (\mathsf{Dec}(\mathcal{CT}^*) \neq M \vee (M, \hat{\Phi}^*, w) \notin \mathcal{R})] = negl(\lambda).\end{aligned}$$

Rerandomizability: The rerandomizability is extended from Π_{RR}, to include π as follows: for all M and $\pi, \mathcal{CT}$ in the support of $\mathsf{Enc}(CRS, PK, M, \hat{\Phi}; w)$, the distribution of $\mathsf{Rerandomize}(PK, \pi, \mathcal{CT})$ is identical to another round of $\mathsf{Enc}(CRS, PK, M, \hat{\Phi}; w)$.

Perfect Decryption Soundness: Equivalent to the perfect decryption soundness in Π_{VD}.

Perfect Zero-Knowledge: Equivalent to the perfect zero-knowledge in Π_{snark}.

5 Proposed Scheme

In this section, we represent the formal construction of the proposed SAVER: Snark-friendly, Additively-homomorphic, and Verifiable Encryption and decryption with Rerandomization. In section 5.1, we provide some intuitive ideas on designing the SAVER. Then we show the construction in section 5.2.

5.1 Main Idea

Before presenting the construction, we provide some intuitive ideas on designing the proposed SAVER. For the voting application in section 1.1, the main objective is to design universal verifiable encryption with additional functionalities: additive-homomorphism, rerandomizability, and verifiable decryption. A naive approach to achieve this is to include the entire encryption algorithm in the zk-SNARK circuit along with the universal relation (to ensure the consistency

Algorithm 1 Encryption-in-the-circuit

$\mathsf{relation}_{\mathsf{enc}}(PK, \mathcal{CT}, \phi_{n+1}, \ldots, \phi_l; M)$:
  $\mathcal{CT} \leftarrow \Pi_{\mathsf{RR,AH}}.\mathsf{Enc}(PK, M)$
  ...

$\mathsf{relation}_{\mathsf{rerand}}(PK, \mathcal{CT}', \phi_{n+1}, \ldots, \phi_l; \pi, \mathcal{CT})$:
  $\Pi_{\mathsf{snark}}.\mathsf{Verify}(\pi, PK, \mathcal{CT}, \phi_{n+1}, \ldots, \phi_l)$
  $\mathcal{CT}' \leftarrow \Pi_{\mathsf{RR,AH}}.\mathsf{Rerandomize}(PK, \mathcal{CT})$

$\mathsf{relation}_{\mathsf{dec}}(\mathcal{CT}, M; SK)$
  $M \leftarrow \Pi_{\mathsf{RR,AH}}.\mathsf{Dec}(SK, \mathcal{CT})$

of m between Prove and Enc), which we refer to as encryption-in-the-circuit method [KZM+15a, KZM+15b].

Algorithm 1 represents zk-SNARK relations required when applying the encryption-in-the-circuit approach. We need three individual relations of $\mathsf{relation}_{\mathsf{enc}}$, $\mathsf{relation}_{\mathsf{rerand}}$, and $\mathsf{relation}_{\mathsf{dec}}$ to satisfy the desired properties. In $\mathsf{relation}_{\mathsf{enc}}$, a rerandomizable homomorphic encryption $\Pi_{\mathsf{RR,AH}}$ like Paillier [Pai99] is combined with the arbitrary relation to satisfy the verifiable additively-homomorphic encryption. In $\mathsf{relation}_{\mathsf{rerand}}$ for rerandomizability, the relation includes the verification of proof π to check the relation of $\mathcal{CT}$, along with the rerandomization of the ciphertext. For example, in the voting application, the administrator must first verify the vote before rerandomizing it, to check that the vote is generated honestly from an eligible user. In $\mathsf{relation}_{\mathsf{dec}}$, the decryption algorithm is included to provide verifiable decryption property. When proceeding the verifiable encryption with these relations, the construction becomes very inefficient: Enc should include $\Pi_{\mathsf{snark}}.\mathsf{Prove}(\mathsf{relation}_{\mathsf{enc}})$, Rerandomize should include $\Pi_{\mathsf{snark}}.\mathsf{Prove}(\mathsf{relation}_{\mathsf{rerand}})$, and Dec should include $\Pi_{\mathsf{snark}}.\mathsf{Prove}(\mathsf{relation}_{\mathsf{dec}})$.

To avoid the inefficiency, we separate encryption from the zk-SNARK relation and provide connectivity between them, similar to the Hash&Prove [FFG+16] or Commit&Prove in LegoSNARK [CFQ19]. Naively binding the encryption and zk-SNARK via commitments as in [FFG+16] may require additional verifications for the linkage. Instead of verifying the linkage separately, we let the ciphertext blend into the original zk-SNARK verification, by replacing the statement (Inputs/Outputs). Intuitively, since zk-SNARK statements are constructed as linear encodings, it is possible to extend the statement as an ElGamal ciphertext.

Let us observe the zk-SNARK verification in [Gro16] as follows:

$$e(A, B) = e(G^{\alpha}, H^{\beta}) \cdot e(\prod_{i=0}^{l} G_i^{\phi_i}, H^{\gamma}) \cdot e(C, H^{\delta})$$

In the equation, $(\phi_1, \ldots, \phi_l)$ can be not only a statement and but also a plaintext. Suppose that ϕ_1 should be encrypted. Let a plaintext message $M = \phi_1$. Then we may construct a ciphertext $\mathcal{CT} = G_1^M$ similar to the ElGamal encryption, which maintains the original verification format as following:

$$e(A,B) = e(G^{\alpha}, H^{\beta}) \cdot e(\mathcal{CT} \cdot \prod_{i=2}^{l} G_i^{\phi_i}, H^{\gamma}) \cdot e(C, H^{\delta})$$

However, it is obvious that $\mathcal{CT}$ should include additional blinding factors mixed to G_1^M. When we denote the blinding factor as X^r, i.e., $\mathcal{CT} = X^r \cdot G_1^M$, the pairing $e(X^r \cdot G_1^M \cdot \prod_{i=2}^{l} G_i^{\phi_i}, H^{\gamma})$ generates unintended γr term in $e(X^r, H^{\gamma})$, which breaks equality of the equation. To resolve this problem, we include $G^{-\gamma}$ in the CRS. The prover modifies the proof element C as $C = C \cdot G^{-\gamma r}$ so that the γr term can be eliminated with respect to the δ from $e(C, H^{\delta})$. As a result, the verification of zk-SNARK can ensure the existence of M in the ciphertext, as well as the soundness of M within the relation.

Another interesting fact is that the form of G_i^M can be plugged into the additive-homomorphism based on the ElGamal encryption. As introduced in [CGS97], it is easy to transform the ElGamal encryption by encrypting G_i^M instead of M, to achieve additive-homomorphism as $G_i^{M_1} \cdot G_i^{M_2} = G_i^{M_1+M_2}$. In this case, the decryption requires *finding the short discrete log* of G_i^M, which restricts the message space to be short enough. Therefore, we split the message M into short message spaces as $M = (m_1 || \dots || m_n)$ (e.g. $|m_i| = 4bits$), and encrypt each block m_i in the form of $X_i^r \cdot G_i^{m_i}$ where X_i^r is a blinding factor. The decryptor who can remove the blinding factor can obtain m_i by the simple brute-forcing (less than 2^4 for $|m_i| = 4bits$).

5.2 SAVER Construction

We now represent a formal construction of the proposed SAVER: Snark-friendly, Additively-homomorphic, and Verifiable Encryption and decryption with Rerandomization. The SAVER utilizes a zk-SNARK Π_{snark} as a building block; we used Groth's protocol [Gro16] as a standard. It is possible to adopt other pairing-based zk-SNARKs such as [GM17] and [KLO19], with some adjustments on Verify_Enc and Rerandomize to assemble the verification and proof format[6].

In the SAVER, a message M is split into n blocks as $M = (m_1 || \cdots || m_n)$, to form a vector $M = \{m_1, \dots, m_n\}$. A ciphertext $\mathcal{CT}$ consists of $n+2$ blocks as $\mathcal{CT} = \{c_0, \cdots, c_n, \psi\}$, where c_0 contains the random, ψ contains an encryption proof, and the remaining c_i contains an encryption of m_i for $1 \le i \le n$. Within the construction, we work with $\{m_1, \dots, m_n\}$, assuming that M is already parsed to $M = (m_1 || \cdots || m_n)$.

Algorithm 2 represents a formal construction of the SAVER. The term relation denotes an arbitrary relation $\mathcal{R}$ for the zk-SNARK, and the terms of α, β, γ, and δ within the functions come from CRS (common reference string) of the adopted zk-SNARK scheme [Gro16].

The SAVER receives any relation which consists of two I/O statements. Statements $m_1, \dots, m_n$ will be encrypted while statements $\phi_{n+1}, \dots, \phi_l$ will be used

[6] Rerandomization of the proof can be viewed as a manipulation, which is prohibited in the simulation-extractable zk-SNARKs. Providing additional terms can resolve this by allowing one-time rerandomization in a restricted manner.

Algorithm 2 SAVER construction

$\mathsf{relation}(m_1, \ldots, m_n, \phi_{n+1}, \ldots, \phi_l; w)$:

$\ldots$

$\mathsf{Setup}(\mathsf{relation})$:

$\hat{CRS} \leftarrow \Pi_{\mathsf{snark}}.\mathsf{Setup}(\mathsf{relation})$
$CRS \leftarrow \hat{CRS} \cup \{G^{-\gamma}\}$
return CRS

$\mathsf{KeyGen}(CRS)$:

$\{s_i\}_{i=1}^n, \{v_i\}_{i=1}^n, \{t_i\}_{i=0}^n, \rho \xleftarrow{\$} \mathbb{Z}_p^*$
$PK \leftarrow (G^{\delta}, \{G^{\delta s_i}\}_{i=1}^n, \{G_i^{t_i}\}_{i=1}^n, \{H^{t_i}\}_{i=0}^n, G^{\delta t_0} \prod_{j=1}^n G^{\delta t_j s_j}, G^{-\gamma \cdot (1+\sum_{j=1}^n s_j)})$
$SK \leftarrow \rho$
$VK \leftarrow (H^{\rho}, \{H^{s_i v_i}\}_{i=1}^n, \{H^{\rho v_i}\}_{i=1}^n)$
return (SK, PK, VK)

$\mathsf{Enc}(CRS, PK, m_1, \ldots, m_n, \phi_{n+1}, \ldots, \phi_l; w)$:

let $PK = (X_0, \{X_i\}_{i=1}^n, \{Y_i\}_{i=1}^n, \{Z_i\}_{i=0}^n, P_1, P_2)$
$r \xleftarrow{\$} \mathbb{Z}_p^*$
$\mathcal{CT} = (X_0^r, X_1^r G_1^{m_1}, \ldots, X_n^r G_n^{m_n}, \psi = P_1^r \cdot \prod_{j=1}^n Y_j^{m_j})$
$\hat{\pi} = (A, B, C) \leftarrow \Pi_{\mathsf{snark}}.\mathsf{Prove}(CRS, m_1, \ldots, m_n, \phi_{n+1}, \ldots, \phi_l; w)$
$\pi \leftarrow (A, B, C \cdot P_2^r)$
return $(\pi, \mathcal{CT})$

$\mathsf{Rerandomize}(PK, \pi, \mathcal{CT})$:

parse $\pi = (A, B, C)$ and $\mathcal{CT} = (c_0, \ldots, c_n, \psi)$
let $PK = (X_0, \{X_i\}_{i=1}^n, \{Y_i\}_{i=1}^n, \{Z_i\}_{i=0}^n, P_1, P_2)$
$r', z_1, z_2 \xleftarrow{\$} \mathbb{Z}_p^*$
$\mathcal{CT}' \leftarrow (c_0 \cdot X_0^{r'}, \ldots, c_n \cdot X_n^{r'}, \psi \cdot P_1^{r'})$
$\pi' \leftarrow (A^{z_1}, B^{z_1^{-1}} \cdot H^{\delta \cdot z_2}, C \cdot A^{z_1 z_2} \cdot P_2^{r'})$
return $(\pi', \mathcal{CT}')$

$\mathsf{Verify\_Enc}(CRS, \pi, \mathcal{CT}, \phi_{n+1}, \cdots, \phi_l)$:

parse $\pi = (A, B, C)$ and $\mathcal{CT} = (c_0, \ldots, c_n, \psi)$
assert $\prod_{j=0}^n e(c_j, H^{t_j}) = e(\psi, H)$
assert $e(A, B) = e(G^{\alpha}, H^{\beta}) \cdot e(\prod_{i=0}^n c_i \cdot \prod_{i=n+1}^l G_i^{\phi_i}, H^{\gamma}) \cdot e(C, H^{\delta})$

as normal I/O statements in plaintext. For the given relation, Setup generates CRS using the adopted zk-SNARKs scheme, with additional $G^{-\gamma}$. KeyGen generates a private key, a public key, and a verification key. Enc encrypts messages $m_1, \ldots, m_n$ and generates a proof π of statement $\Phi = (m_1, \ldots, m_n, \phi_{n+1}, \ldots, \phi_l)$. To check the truth of statement Φ, Verify_Enc takes π and $\mathcal{CT}$ as inputs for verification. Rerandomize does rerandomization of the given ciphertext and the proof. Note that the rerandomized proof is a valid proof of the statement. Dec decrypts the ciphertext $\mathcal{CT}$ by performing decryption for each block $c_1, \ldots, c_n$, to output $m_1, \ldots, m_n$ and a decryption proof ν. The original message M can be restored

$\mathsf{Dec}(CRS, SK, VK, \mathcal{CT})$:
parse $SK = \rho$, $VK = (V_0, \{V_i\}_{i=1}^{n}, \{V_i\}_{i=n+1}^{2n})$, and $\mathcal{CT} = (c_0, \ldots, c_n, \psi)$
for $i = 1$ **do** to n
$\frac{e(c_i, V_{n+i})}{e(c_0, V_i)^{\rho}} = e(G_i, V_{n+i})^{m_i}$
compute a discrete log of $e(G_i, V_{n+i})^{m_i}$ to obtain m_i
end for
$\nu \leftarrow c_0^{\rho}$
return $(m_1, \ldots m_n, \nu)$

$\mathsf{Verify\_Dec}(CRS, VK, m_1, \ldots m_n, \nu, \mathcal{CT})$:
parse $VK = (V_0, \{V_i\}_{i=1}^{n}, \{V_i\}_{i=n+1}^{2n})$ and $\mathcal{CT} = (c_0, \ldots, c_n, \psi)$
assert $e(\nu, H) = e(c_0, V_0)$
for $i = 1$ **do** to n
assert $\frac{e(c_i, V_{n+i})}{e(\nu, V_i)} = e(G_i, V_{n+i})^{m_i}$
end for

as $M = (m_1 || \ldots || m_n)$. The honest decryption of $\mathcal{CT}$ can be proved by calling $\mathsf{Verify\_Dec}$ with a message M and a decryption proof ν.

The ciphertext $\mathcal{CT}$ in SAVER satisfies additive-homomorphic property. Given $\mathcal{CT} = (X_0^r, \{X_i^r G_i^{m_i}\}_{i=1}^{n}, P_1^r \prod_{j=1}^{n} Y_j^{m_j})$ and $\mathcal{CT}' = (X_0^{r'}, \{X_i^{r'} G_i^{m_i'}\}_{i=1}^{n}, P_1^{r'} \prod_{j=1}^{n} Y_j^{m_j'})$, it is easy to see that $\mathcal{CT} \cdot \mathcal{CT}' = (X_0^{r+r'}, \{X_i^{r+r'} G_i^{m_i+m_i'}\}_{i=1}^{n}, P_1^{r+r'} \prod_{j=1}^{n} Y_j^{m_j+m_j'})$, which satisfies additive-homomorphism.

6 Security Proof

To satisfy the definition of SAVER, the scheme should satisfy completeness, indistinguishability, encryption knowledge soundness, rerandomizability, and perfect zero-knowledge. The completeness is easy to verify in algorithm 2. For the perfect zero-knowledge, it is sufficient to show that the proof π in SAVER maintains the perfect zero-knowledge of zk-SNARK [Gro16].

Lemma 2. *The proof π generated in SAVER is within the same distribution from the proof $\hat{\pi}$ of Π_{snark}.*

Since $\hat{\pi}$ is in a random distribution and P_2^r is in a random distribution from r, $C \cdot P_2^r$ is also within a same random distribution.

6.1 Indistinguishability

In this section, we prove the standard IND-CPA security of our SAVER.

Theorem 1. *Suppose the Decisional $(d(\lambda), q(\lambda))$-Poly assumption holds in $\mathcal{BG}$. Then SAVER is IND-CPA secure.*

Suppose that $\mathcal{A}$ has an advantage ϵ in attacking the SAVER. Using $\mathcal{A}$, we build an algorithm $\mathcal{B}$ that solves the D-Poly problem in $\mathcal{BG}$. We first describe the overall sketch of our proof as follows.

The game starts with selecting the generator G, H and the D-Poly secret vector $\boldsymbol{x} = \{\alpha, \beta, \gamma, \delta, x, t_0, \ldots, t_n, s_1, \ldots, s_n, v_1, \ldots, v_n, \rho\}$ from $\mathbb{Z}_p^*$. As a challenger in the D-Poly game, algorithm $\mathcal{B}$ can query polynomials $g_i(X_1, \cdots, X_q)$ and $h_j(X_1, \cdots, X_q)$ to the oracles $\mathcal{O}^1_{G,\boldsymbol{x}}$ and $\mathcal{O}^2_{H,\boldsymbol{x}}$ to receive corresponding $G^{g_i(\boldsymbol{x})}$ and $H^{h_j(\boldsymbol{x})}$, within a polynomial time.

With the help of these oracles, $\mathcal{B}$ simulates the encryption oracle for $\mathcal{A}$'s encryption queries; $\mathcal{B}$ receives query M_i from $\mathcal{A}$ within the polynomial time to return corresponding ciphertext, proof and its decryption proof as $(\mathcal{CT}_i, \pi_i, \nu_i)$.

Then for the challenge, $\mathcal{B}$ outputs $g_c(X_1, \cdots, X_q)$ which satisfies $g_c(x) \notin [\![Q_1]\!] \times [\![Q_2]\!] \times [\![Q_2]\!]^{-1}$, to receive $T = T_b$ from the D-Poly game where T_b is randomly chosen from $T_1 = G^{g_c(\boldsymbol{x})}$ or $T_0 \xleftarrow{\$} \mathbb{G}_1$. The goal of algorithm $\mathcal{B}$ is to guess b, outputting $b' = 1$ if the T is generated from $G^{g_c(\boldsymbol{x})}$ and $b' = 0$ otherwise. Algorithm $\mathcal{B}$ works by interacting with $\mathcal{A}$ in an IND-CPA game as follows:

Setup: To generate the CRS, $\mathcal{B}$ runs a Setup(relation) in [Gro16] with using D-Poly oracles. By querying $g_i(X_1, \cdots, X_q)$ or to the corresponding oracle $\mathcal{O}^1_{G,\boldsymbol{x}}$ or $\mathcal{O}^2_{H,\boldsymbol{x}}$, $\mathcal{B}$ can generate all CRS parameters $(G^\alpha, G^\beta, G^\delta, \cdots)$ without the knowledge of the secret vector $\boldsymbol{x}$.

KeyGen: Algorithm $\mathcal{B}$ can run the original KeyGen(CRS) by utilizing the existing CRS generated from above. $\mathcal{B}$ returns (PK, VK) to initialize $\mathcal{A}$. Additionally, $\mathcal{B}$ generates the tag key $\hat{\nu} = G^{\delta\rho}$ by querying $\delta\rho$ to $\mathcal{O}^1_{G,\boldsymbol{x}}$.

Enc1: After the initialization, $\mathcal{A}$ may query $\mathcal{B}$ for the encryption of the message $M_i = (m_1 || \cdots || m_n)$, to obtain the corresponding ciphertext and decryption proof $\mathcal{CT}_i, \nu_i$. Note that $\mathcal{A}$ can generate $\mathcal{CT}_i$ itself with using PK (as in standard IND-CPA game); the purpose of encryption query is for the additional information ν_i which $\mathcal{A}$ cannot create itself. For $\mathcal{A}$'s query M_i, $\mathcal{B}$ generates a ciphertext $\mathcal{CT}_i$ by calling $\mathsf{Enc}(CRS, PK, M_i)$ with picking fresh random r_i, and creates an encryption proof π_i by calling $SimProve_{snark}(CRS, m_1, \ldots, m_n, \phi_{n+1}, \ldots, \phi_l)$ with given statement $(m_1, \ldots, m_n, \phi_{n+1}, \ldots, \phi_l)$ where $SimProve_{snark}$ generates a simulated proof available in every zk-SNARK scheme since the zk-SNARK scheme is zero knowledge. Then, $\mathcal{B}$ crafts the decryption proof $\nu_i = \hat{\nu}^{r_i}$, and returns $\pi_i, \mathcal{CT}_i, \nu_i$ as a response to $\mathcal{A}$.

Challenge: When $\mathcal{A}$ outputs M_0 and M_1 for the IND-CPA challenge, $\mathcal{B}$ picks $b \in \{0, 1\}$ for M_b then challenges the D-Poly game to receive T and create the ciphertext $\mathcal{CT}$ by implicitly setting $r = x^{d+1} \cdot r'$ ($r' \xleftarrow{\$} \mathbb{Z}_p^*$). To describe $\mathcal{B}$'s response on $M_b = (m_1 || \ldots || m_n)$, we first define two events on generating $\mathcal{CT} = (c_0, \ldots, \psi)$: REAL and FAKE. Among the blocks $c_1, \ldots, c_n$ which are supposed to contain the encrypted message (excluding c_0 and ψ which are not related to the message), two events are defined for each block c_i as follows:

1. REAL: The block c_i is crafted honestly with a *real* message as $G^{\delta s_i \cdot r} G_i^{m_i}$, by querying $g(\boldsymbol{x})$ to $\mathcal{O}^1_{G,\boldsymbol{x}}$.
2. FAKE: The block c_i is crafted with a *random* message $\mu_i \xleftarrow{\$} \mathbb{Z}_p^*$ as $G^{\delta s_i \cdot r} G_i^{\mu_i}$, by querying $g(\boldsymbol{x})$ to $\mathcal{O}^1_{G,\boldsymbol{x}}$.

When creating $c_1, \ldots, c_n$, $\mathcal{B}$ picks $j \in \{1, \ldots, n\}$ to use the challenge response T in c_j, and let $c_1, \ldots, c_{j-1}$ generated from REAL while $c_{j+1}, \ldots, c_n$ are generated from FAKE. $\mathcal{B}$ gains advantage of winning the game only when $\mathcal{A}$ guesses b *exactly* from the challenge. If $\mathcal{A}$ can already distinguish b without the challenge c_j, the game fails because $\mathcal{A}$ will always distinguish b regardless of the nature of T. On the other hand, if $\mathcal{A}$ requires $c_{j'}$ for $j' > j$ to distinguish b, the game fails because $\mathcal{A}$ always fails to distinguish b regardless of the nature of T since $c_{j'}$ is from FAKE. More specifically, from $\mathcal{A}$'s view, there exists $j' \in \{1, \ldots, n\}$ where $\mathcal{A}$ cannot distinguish b when $c_1, \ldots, c_{j'-1}$ are from REAL, but can distinguish b when $c_1, \ldots, c_{j'}$ are from REAL. Therefore, by choosing j, $\mathcal{B}$ is *guessing* j'; if $\mathcal{B}$'s guess is correct, i.e., $j = j'$ with the probability of $\frac{1}{n}$, $\mathcal{B}$ can win the D-Poly game since $\mathcal{A}$ works differently depending on the nature of T.

To prepare the challenge, $\mathcal{B}$ picks $r' \xleftarrow{\$} \mathbb{Z}_p^*$ and interacts with the D-Poly oracle $\mathcal{O}^1_{G,\boldsymbol{x}}$ by implicitly setting r as $x^{d+1} \cdot r'$. $\mathcal{B}$ first queries $\delta \cdot x^{d+1}$ to receive $\hat{c_0}$. Next, $\mathcal{B}$ prepares the random parts for all the blocks except j-th block by querying $\{g_i(x) = \delta s_i \cdot x^{d+1}\}_{i=1,\neq j}^n$ to receive $\{\hat{c_i}\}_{i=1,\neq j}^n$. Then $\mathcal{B}$ prepares the ingredient for ψ, by querying $x^{d+1} \cdot (\delta t_0 \sum_{i=1}^n \delta t_i s_i)$ to receive $\hat{\psi_1}$ and querying $\sum_{i=1}^j y_i(x) \cdot t_i m_i + \sum_{i=j+1}^n y_i(x) \cdot t_i \mu_i$ to receive $\hat{\psi_2}$. For the encryption proof π, $\mathcal{B}$ picks $A, B \leftarrow \mathbb{Z}_p^*$ and queries the simulation equation (e.g. $\frac{AB - \alpha\beta - \sum_{i=0}^l a_i y_i}{\delta}$ in [Gro16]), to obtain C and generate the tuple of simulated proof $\pi = (A, B, C)$. Note that $\mathcal{B}$ is required to simulate the proof since it does not know the random r, which is a witness for the relation.

When $\{\hat{c_i}\}_{i=1,\neq j}^n, \hat{\psi_1}, \hat{\psi_2}$ and π are ready, $\mathcal{B}$ outputs a challenge query for the j-th block $g(x) = \delta s_j \cdot x^{d+1}$ to receive T. Notice that the challenge query $g(x)$ satisfies $g(x) \notin [\![Q_1]\!] \times [\![Q_2]\!] \times [\![Q_2]\!]^{-1}$, since s_j is independent. Then $\mathcal{B}$ generates the ciphertext $\mathcal{CT}$ for $M_b = (m_1 || \ldots || m_n)$ by exploiting the received elements as $c_0 = \hat{c_0}^{r'}$, $\{c_i = \hat{c_i}^{r'} G_i^{m_i}\}_{i=1}^{j-1}, c_j = T^{r'} G_j^{m_j}$ (REAL), and $\{c_i = \hat{c_i}^{r'} G_i^{\mu_i}\}_{i=j+1}^n$ (FAKE). Finally, $\mathcal{B}$ computes ψ by computing as $\psi = \hat{\psi_1}^{r'} \cdot \hat{\psi_2}$, and returns $\mathcal{CT} = \{c_i\}_{i=1}^n, \psi$ and π to $\mathcal{A}$.

Enc2: $\mathcal{B}$ can respond to $\mathcal{A}$'s queries same as Enc1.

Guess: Finally, $\mathcal{A}$ outputs a guess $b' \in \{0, 1\}$. Algorithm $\mathcal{B}$ concludes its own game by outputting a guess as follows. If $b = b'$ then $\mathcal{B}$ outputs 1 meaning $T = G^{g_c(\boldsymbol{x})}$. Otherwise, it outputs 0 meaning T is random in $\mathbb{G}_T$.

When the input tuple is sampled from $T_1 = G^{g_c(\boldsymbol{x})}$, and $\mathcal{B}$'s guess with the probability of $\frac{1}{n}$ is correct as $j = j'$, then $\mathcal{A}$'s view is identical to its view in a real attack game and therefore $\mathcal{A}$ satisfies $\frac{1}{n} \cdot |Pr[b = b'] - 1/2| \geq \epsilon$. When the

input tuple is sampled from $T_0 \xleftarrow{\$} \mathbb{G}_1$, then $Pr[b = b'] = 1/2$. Therefore, with G, H uniform in $\mathcal{BG}$, $\boldsymbol{x}$ uniform in $\mathbb{Z}_p^*$, and T uniform in $\mathbb{G}_T$ we have that

$$\begin{aligned}&|Pr[\mathcal{B}(\mathcal{BG}, q_i(\boldsymbol{x}), G^{g_c(\boldsymbol{x})}) = 0] \\ &- Pr[\mathcal{B}(\mathcal{BG}, q_i(\boldsymbol{x}), T) = 0]| \geq |(1/2 + n\epsilon) - 1/2| = n\epsilon\end{aligned}$$

as required, which completes the proof of the theorem.

6.2 Encryption Soundness

In this section, we prove the soundness of π and $\mathcal{CT}$ in $\mathsf{Verify\_Enc}$, indicating that the M which is encrypted to $\mathcal{CT}$ is indeed included in the I/O of the conjoined pairing-based SNARK [Gro16]. Formally, we show that the probability of any adversary forging $(\pi^*, \mathcal{CT}^*, \hat{\Phi}^*)$ where $\hat{\Phi}^* = \{\phi_{n+1}, \cdots, \phi_l\}$ which passes the $Verify\_Enc$ but $Dec(\mathcal{CT}^*) \neq M$ or $(M, \hat{\Phi}^*, w) \notin \mathcal{R}$ is negligible.

Theorem 2. *Suppose the batch − PKE assumption holds, and the soundness of conjoined pairing-based zk-SNARK [Gro16] holds. Then* SAVER *satisfies the encryption knowledge soundness.*

To prove the theorem, we show that any adversary which breaks the soundness of the SAVER can break the *batch-PKE* assumption or *SNARK-snd*, i.e., soundness of the conjoined SNARK [Gro16]. Formally, for all PPT adversaries $\mathcal{A}$ there exists a PPT algorithm $\mathcal{B}, \mathcal{C}$ and a PPT extractor $\chi_{\mathcal{B}}$ such that

$$\begin{aligned}&\mathbf{Adv}^{sound}_{\mathsf{SAVER},\mathcal{A}}(\lambda) = Pr[(CRS, \tau) \leftarrow \mathsf{Setup}(\mathcal{R}), (PK, SK, VK) \leftarrow \mathsf{KeyGen}(CRS), \\ &(\pi^*, \mathcal{CT}^*, \hat{\Phi}^*) \leftarrow \mathcal{A}(CRS, PK, VK), (M, w) \leftarrow \chi_{\mathcal{A}}(trans_{\mathcal{A}}) : \\ &\mathsf{Verify\_Enc}(CRS, \pi^*, \mathcal{CT}^*, \hat{\Phi}^*) = 1 \wedge (\mathsf{Dec}(\mathcal{CT}^*) \neq M \vee (M, \hat{\Phi}^*, w) \notin \mathcal{R})] \\ &\leq \mathbf{Adv}^{batch\text{-}PKE}_{\mathcal{R},\mathcal{B},\chi_{\mathcal{B}}}(\lambda) + \mathbf{Adv}^{sound}_{\Pi_{\mathsf{snark}},\mathcal{C},\chi_{\mathcal{C}}}(\lambda).\end{aligned}$$

First, since SAVER requires additional $G^{-\gamma}$ in the CRS, it is necessary to assure that the soundness of the zk-SNARK [Gro16] still holds with the extended CRS. Fortunately, this issue is resolved instantly from the fact that the security proof in [Gro16] also considers γ term, according to the affine prover strategy. In the statistical knowledge soundness of [Gro16], the element A is demonstrated as:

$$A = A_\alpha \alpha + A_\beta \beta + A_\gamma \gamma + a_\delta \delta A(x) + \sum_{i=0}^{l} A_i \frac{y_i(x)}{\gamma} + \sum_{i=l+1}^{m} A_i \frac{y_i(x)}{\delta} + A_h(x) \frac{t(x)}{\delta}$$

Observe that the $A_\gamma \gamma$ is included, indicating that the G^γ is within the consideration of ingredients. Since the A_γ term is eliminated in the proof, adding $G^{-\gamma}$ in the CRS of [Gro16] does not affect the soundness of the SNARK. Similar to [Gro16], we now view the verification equations as an equality of multi-variate

Laurent polynomials. By the Schwartz-Zippel lemma the prover has negligible success probability unless both verification equations hold.

Since $\pi^*, \mathcal{CT}^*, \hat{\Phi}^*$ passes the verification, it passes the two equations in Verify_Enc as stated below:

$$e(c_1, H^{t_1}) \times \cdots \times e(c_n, H^{t_n}) = e(\psi, H) \tag{1}$$

$$e(A, B) = e(G^{\alpha}, H^{\beta}) \times e(\prod_{i=0}^{n} c_i \cdot \prod_{i=n+1}^{l} G_i^{\phi_i}, H^{\gamma}) \times e(C, H^{\delta}) \tag{2}$$

When we see the equation 1, there always exists t_i, since they are fixed in the expression itself as H^{t_i}. Therefore, ψ must consist of $y_i(x)t_i$ and $\delta t_0 + \sum_{j=1}^{n} \delta t_j s_j$ since the only terms which include t_i in the CRS and PK are $G_1^{t_1}, \cdots, G_n^{t_n}$ and $G^{\delta t_0 + \sum_{j=1}^{n} \delta t_j s_j}$. Let us express auxiliary indeterminate for each variable as X, which is yet ambiguous. Then, the exponents linearly satisfy the equation below:

$$\begin{aligned} &Xt_0 + Xt_1 + \cdots + Xt_n = \\ &\qquad Xy_1(x) \cdot t_1 + \cdots + Xy_n(x) \cdot t_n + X(\delta t_0 + \sum_{j=1}^{n} \delta t_j s_j) \end{aligned} \tag{3}$$

When observing equation 3 above, note that the terms with $y_i(x) \cdot t_i$ and $\delta t_0 + \sum_{j=1}^{n} \delta t_j s_j$ must both exist, since they are the only terms which can balance the $t_1, \cdots, t_n$ and t_0 on the left of equal sign. Then, to meet the terms with $y_1(x), \cdots, y_n(x)$, there must also exist $y_i(x)$ in each term with $t_1, \cdots, t_n$ on the left of equal sign. For the unknown coefficients η_i', this leads to:

$$\begin{aligned} &Xt_0 + (X + \eta_1' y_1(x)) \cdot t_1 + \cdots + (X + \eta_n' y_n(x)) \cdot t_n = \\ &\qquad Xy_1(x) \cdot t_1 + \cdots + Xy_n(x) \cdot t_n + X(\delta t_0 + \sum_{j=1}^{n} \delta t_j s_j) \end{aligned} \tag{4}$$

Since only $\delta t_0 + \sum_{j=1}^{n} \delta t_j s_j$ includes δt_0, the t_0 term on the left must only include δ to generate δt_0. Finally, there remains $\delta t_j s_j$ in $\sum_{j=1}^{n} \delta t_j s_j$; X in each t_j term must be related to δs_j to generate $\delta t_j s_j$. For the unknown coefficients η_0' and η_i'', this leads to:

$$\begin{aligned} &(\eta_0' \delta) t_0 + (\eta_1' \delta s_1 + \eta_1'' y_1(x)) t_1 + \cdots + (\eta_n' \delta s_n + \eta_n'' y_n(x)) t_n = \\ &\qquad Xy_1(x) \cdot t_1 + \cdots + Xy_n(x) \cdot t_n + X(\delta t_0 + \sum_{j=1}^{n} \delta t_j s_j) \end{aligned} \tag{5}$$

Now we can complete the equation with filling up each auxiliary X on the right side with unknown coefficients $\eta_0', \{\eta_i', \eta_i''\}_{i=1}^{n}$. Especially, since the term with $\delta t_0 + \sum_{j=1}^{n} \delta t_j s_j$ is unique, the coefficients for δt_0 and $\delta s_i t_i$ (i.e. $\eta_0', \cdots, \eta_n'$) must be same as η'. Therefore, for the unknown coefficients η' and η_i'', the equation can be arranged as:

$$(\eta'\delta)t_0 + (\eta'\delta s_1 + \eta''_1 y_1(x))t_1 + \cdots + (\eta'\delta s_n + \eta''_n y_n(x))t_n = \eta''_1 y_1(x)\cdot t_1 + \cdots + \eta''_n y_n(x)\cdot t_n + \eta'(\delta t_0 + \sum_{j=1}^{n} \delta t_j s_j) \tag{6}$$

When representing the coefficients of $t_0, \cdots, t_n$ on the left as $a_0, \cdots, a_n$ (i.e. $a_0t_0 + a_1t_1 + \cdots + a_nt_n$), each a_i can be viewed as a value from a multi-variate polynomial $f_i(\boldsymbol{x})$ which consists of coefficients η'_0 and η''_i. Let us represent C from π^* as G^b. Putting this into the verifying equation 1 gives us:

$$e(G^{a_1}, H^{t_0}) \times \cdots \times e(G^{a_n}, H^{t_n}) = e(G^b, H) \tag{7}$$

Observe that equation 6 above is equivalent to the check in *batch-PKE* where t_i corresponds to α_i: therefore there exists an extractor $\chi_{\mathcal{B}}$ which can extract all the coefficients η'_0 and η''_i from $a_i = f_i(\boldsymbol{x})$. With the knowledge of η'_0, η''_i in equation 6, it is obvious that η' is equivalent to r and η''_i is equivalent to m_i, since the equation is in the same form as the original scheme with $c_0 = G^{\delta r}, c_1 = G^{\delta s_1 r} \cdot G^{y_1(x)m_1}, \cdots, c_n = G^{\delta s_n r} \cdot G^{y_n(x)\cdot m_n}$. If $Dec(\mathcal{CT}^*) \neq M$, then there exists $m_i^* = \eta''_i \neq m_i$; the extractor failed as $\eta''_i \neq [A(\tau)]$ which breaks the *batch-PKE*.

$$\therefore Pr[Dec(\mathcal{CT}^*) \neq M] = \mathbf{Adv}^{batch\text{-}PKE}_{\mathcal{R},\mathcal{B},\chi_{\mathcal{B}}}(\lambda) \tag{8}$$

The remaining case is where $(M, \hat{\Phi}^*, w) \notin \mathcal{R}$. In this case, we start with the fact that $\pi^*, \mathcal{CT}^*, \hat{\Phi}^*$ passes equation 2, revisited as follows.

$$e(A, B) = e(G^\alpha, H^\beta) \times e(\prod_{i=0}^{n} c_i \cdot \prod_{i=n+1}^{l} G_i^{\phi_i}, H^\gamma) \times e(C, H^\delta) \tag{2}$$

Since equation 8 let $\mathcal{CT}^*$ satisfy $Dec(\mathcal{CT}^*) = M$, we can write $\mathcal{CT}$ as a original form, i.e., $c_0 = G^{\delta r}, c_i = G^{\delta s_i r} \cdot G^{m_i}$. Putting this into equation 2 gives us:

$$e(A, B) = e(G^\alpha, H^\beta) \times e(G^{\delta r} \cdot \prod_{i=1}^{n} (G^{\delta s_i r} \cdot G^{y_i(x)\cdot m_i}) \cdot \prod_{i=n+1}^{l} G_i^{\phi_i}, H^\gamma) \times e(C, H^\delta) \tag{9}$$

Observe that $e(G^{\delta r} \cdot \prod_{i=1}^{n} (G^{\delta s_i r} \cdot G^{y_i(x)\cdot m_i}) \cdot \prod_{i=n+1}^{l} G_i^{\phi_i}, H^\gamma)$ always generates $\gamma\delta s_i$ term. To neutralize $\gamma\delta s_i$, the only possible way is either by also generating $\gamma\delta s_i$ in $e(A, B)$ on the left of equal sign, or by generating the same term in $e(C, H^\delta)$ on the right to eliminate $\gamma\delta s_i$.

Case 1 - generating $\gamma\delta s_i$ in $e(A, B) = e(G^a, H^b)$ on the left:

When considering the term with $\alpha\beta$ which exists in $e(G^\alpha, H^\beta)$ on the right side, a must include α and b must include β, since there are no H^α in the $\mathbb{G}_2$

of CRS, PK, VK. From the fact that there are no δs_i in $\mathbb{G}_2$, the only way to generate the $\gamma\delta s_i$ term is to include δs_i in a and include γ in b as follows:

$$a = X_\alpha \alpha + X_{\delta s_i} \delta s_i + \cdots, \qquad b = X_\beta \beta + X_\gamma \gamma + \cdots$$

However, this let $e(G^a, H^b)$ create $\alpha\gamma$ and $\beta\delta s_i$, which does not exist in equation 9. Therefore, γ cannot exist in a nor b, which indicates that Case 1 cannot exist.

Case 2 - generating $\gamma\delta s_i$ in $e(C, H^\delta) = e(G^c, H^\delta)$ on the right:

The remaining case is where c includes γs_i to generate $\gamma\delta s_i$ in $e(G^c, H^\delta)$ and eliminate the $\gamma\delta s_i$ term. The only term which includes γs_i is $R = G^{-\gamma\cdot(1+\sum_{j=1}^{n} s_j)}$, and therefore c must include $-\gamma(1+\sum_{j=1}^{n} s_j)$. We can write c as $c = \eta\cdot(-\gamma(1+\sum_{j=1}^{n} s_j) + A_x)$, where η is an unknown coefficient, and A_x is a remaining auxiliary polynomial. Putting this into equation 9 gives us:

$$\begin{aligned} e(A,B) = e(G^\alpha, H^\beta) \\ \times e((G^{\delta r} \cdot G^{\sum_{i=1}^{n} \delta s_i r} \cdot G^{\sum_{i=1}^{n} y_i(x)\cdot m_i}) \cdot \prod_{i=n+1}^{l} G_i^{\phi_i}, H^\gamma) \\ \times e(G^{\eta\cdot(-\gamma(1+\sum_{j=1}^{n} s_j))} \cdot G^{A_x}, H^\delta) \end{aligned} \tag{10}$$

To balance $\gamma\delta s_i$, the $\sum_{i=1}^{n} \delta s_i r$ term must meet $\eta\cdot(-\gamma(1+\sum_{j=1}^{n} s_j))$ to cancel out, and therefore $\eta = r$. This leads to:

$$\begin{aligned} e(A,B) = e(G^\alpha, H^\beta) \\ \times e(G^{\sum_{i=1}^{n} y_i(x)\cdot m_i} \cdot G^{\sum i=n+1^{l} y_i(x)\cdot\phi_i}, H^\gamma) \times e(G^{A_x}, H^\delta) \end{aligned} \tag{11}$$

When observing equation 11 above, $G^{\sum_{i=1}^{n} y_i(x)\cdot m_i} \cdot G^{\sum i=n+1^{l} y_i(x)\cdot\phi_i}$ can be combined into $G^{\sum i=1^{n} a_i y_i(x)}$, since $m_1, \cdots, m_n$ and $\phi_{n+1}, \cdots, \phi_l$ are $\Phi^* = \{a_1, \cdots, a_l\}$. This let equation 11 equivalent to the verification of the original pairing-based SNARK [Gro16] for the proof elements $(A, B, C = G^{A_x})$ and Φ^*. If $(M, \hat{\Phi}^*, w) \notin \mathcal{R}$, then there exists m_i or ϕ_i which is not in the relation, but passes the verification of [Gro16]. This breaks the soundness of the SNARK, which concludes the proof as below:

$$\therefore Pr[(M, \hat{\Phi}^*, w) \notin \mathcal{R}] = \mathbf{Adv}^{sound}_{\Pi_{\mathsf{snark}}, \mathcal{C}, \chi_{\mathcal{C}}}(\lambda)$$

6.3 Rerandomizability

In this section, we show that a new rerandomized proof and ciphertext $\pi', \mathcal{CT}'$ takes a same distribution as the original proof and ciphertext $\pi, \mathcal{CT}$ with a fresh random, which can assure the security of rerandomized proofs. The rerandomization of $\mathcal{CT}$ to $\mathcal{CT}'$ is as follows:

$$\mathcal{CT} = (X_0^r, X_1^r G_1^{m_1}, \cdots, X_n^r G_n^{m_n}, P_1^r \prod_{j=1}^{n} Y_j^{m_j})$$

$$r' \xleftarrow{\$} \mathbb{Z}_p^*$$

$$\mathcal{CT}' = (X_0^r \cdot X_0^{r'}, X_1^r G_1^{m_1} \cdot X_1^{r'}, \cdots, X_n^r G_n^{m_n} \cdot X_n^{r'}, P_1^r \prod_{j=1}^{n} Y_j^{m_j} \cdot P_1^{r'})$$

$$\therefore \mathcal{CT}' = (X_0^{r+r'}, X_1^{r+r'} G_1^{m_1}, \cdots, X_n^{r+r'} G_n^{m_n}, P_1^{r+r'} \prod_{j=1}^{n} Y_j^{m_j})$$

It is easy to see that $\mathcal{CT}'$ is a valid ciphertext with a fresh random $r + r'$.

For the rerandomization of $\pi = (A, B, C)$ to $\pi' = (A', B', C')$, it is necessary to show that the original proof and the rerandomized proof are both within a uniform distribution. Let us decompose the proof elements (A, B, C) to (G^a, H^b, G^c) as its original form (random r from SAVER denoted as r^* to avoid the duplication):

$$a = \alpha + \sum_{i=0}^{m} a_i u_i(x) + r\delta \qquad b = \beta + \sum_{i=0}^{m} a_i v_i(x) + s\delta$$

$$c = \frac{\sum_{i=l+1}^{m} a_i y_i + h(x)t(x)}{\delta} + As + Br - rs\delta - \gamma \cdot (1 + \sum_{j=1}^{n} s_j) r^*$$

Observe that the randomness of a depends on r, and the randomness of b depends on s. The randomness of c is determined by a and b; if a and b is generated appropriately, c is automatically determined within a uniform distribution. Therefore, it is sufficient to show that a' and b' from the rerandomized proof are appropriate randoms. When representing (A, B, C) as (G^a, H^b, G^c), the a', b', c' in the rerandomized proof $(G^{a'}, H^{b'}, G^{c'})$ are:

$$a' = a \cdot z_1 \qquad b' = b \cdot z_1^{-1} + \delta \cdot z_2 \qquad c' = c + a \cdot z_1 z_2 - \gamma \cdot (1 + \sum_{j=1}^{n} s_j) r^{*\prime}$$

It is straightforward that a' and b' are within a uniform distribution, where a' depends on a fresh random z_1, and b' depends on a fresh random z_2. Since a' and b' are appropriate randoms, we can conclude that c' is also determined within a uniform distribution.

6.4 Decryption Soundness

In this section, we prove the soundness of the decryption proof ν in Verify_Dec, indicating that there cannot exist any ν^* which is connected to the wrong ciphertext but still passes the Verify_Dec.

Theorem 3. *In our* SAVER *scheme, there cannot exist any* $(M^*, \nu^*, \mathcal{CT}^*)$ *where* ν^* *verifies, but* $\mathsf{Dec}(\mathcal{CT}^*) \neq M^*$.

Let us violate the theorem and assume that there exists $(M^*, \nu^*, \mathcal{CT}^*)$ where ν^* verifies, but $\mathsf{Dec}(\mathcal{CT}^*) \neq M^*$. More specifically, for $\mathcal{CT}^* = (c_0^*, \cdots, \psi^*)$ and $M^* = (m_1^*, \cdots, m_n^*)$ there exists a block c_j^* which is *not* decrypted to m_j^* for any $j \in \{1, \cdots, n\}$ while ν^* passes the verifications in Verify_Dec.

Since the decryption proof ν^* verifies, the 2nd equation of Verify_Dec holds as follows:

$$\frac{e(c_j^*, V_{n+j})}{e(\nu^*, V_j)} = e(G_j, V_{n+j})^{m_j^*} \tag{12}$$

However, since $\mathsf{Dec}(\mathcal{CT}^*) \neq M^*$,

$$\frac{e(c_j^*, V_{n+j})}{e(c_0^*, V_j)^{\rho}} \neq e(G_j, V_{n+j})^{m_j^*} \tag{13}$$

When comparing equations 12 and 13, the only difference between two equations are $e(\nu^*, V_j)$ and $e(c_0^*, V_j)^{\rho}$: therefore $\nu^* \neq (c_0^*)^{\rho}$.

However, this contradicts the first equation of Verify_Dec:

$$\begin{aligned} e(\nu^*, H) &= e(c_0^*, V_0) \\ \therefore \nu^* &= (c_0^*)^{\rho} \end{aligned} \tag{14}$$

Therefore, we conclude that there cannot exist any $(m^*, \nu^*, \mathcal{CT}^*)$ where ν^* verifies and $\mathsf{Dec}(\mathcal{CT}^*) \neq M^*$.

7 Vote-SAVER

We present a formal protocol for the voting system application in section 1.1, named as Vote-SAVER. As described in the scenario, the Vote-SAVER consists of series of interactions between multiple administrators and multiple voters, with utilizing the SAVER in section 5 as a building block. For the additional building blocks, we use the publicly-available $BlockChain$, a collision-resistant hash function H, membership test functions $MerkleTree$, $GetMerklePath$, $GetMerkleRoot$ from Zerocash [BCG+14]. Note that $sn, rt, \boldsymbol{path}$ are also from the membership test, where sn is a serial number, rt is a Merkle root, and $\boldsymbol{path}$ is a vector of co-paths for constructing the Merkle tree. We use ID for each user's identity, and eid to distinguish each individual election.

Algorithm 3 represents a series of functions for the voter's side, algorithm 4 represents a function (possibly smart-contract) for the $BlockChain$ nodes, and algorithm 5 represents functions for the administrator. For the scenario, the election proceeds as follows.

Phase 0: *init system.* Before running the system, the CRS should be generated from InitSystem. To be more accurate, this should be done by a trusted third party or by a general consensus, rather than an individual administrator. Then,

Algorithm 3 Voting system voter

$\mathsf{GenKey}(1^\kappa, ID)$:

$sk \xleftarrow{\$} \{0,1\}^\kappa$
$sk_{ID} \leftarrow ID||sk$
$pk_{ID} = H(sk_{ID})$
return sk_{ID} to *voter*
publish pk_{ID}

$\mathsf{Vote}(CRS, PK_{eid}, eid, rt, \boldsymbol{pklist}, M, pk_{ID}, sk_{ID})$:

$\boldsymbol{path} \leftarrow GetMerklePath(pk_{ID}, \boldsymbol{pklist})$
$sn \leftarrow H(eid \parallel sk_{ID})$
$\pi, \mathcal{CT} \leftarrow \mathsf{SAVER.Enc}(CRS, PK_{eid}, M, eid, sn, rt; \boldsymbol{path}, sk_{ID})$
send $ballot = \{eid, sn, \pi, \mathcal{CT}\}$ to $BlockChain$ network

$\mathsf{VerifyVote}(CRS, \pi'_{ID}, \mathcal{CT}'_{ID}, eid, rt)$:

$sn \leftarrow H(eid \parallel sk_{ID})$
$assert\ \mathsf{SAVER.Verify\_Enc}(CRS, \pi'_{ID}, \mathcal{CT}'_{ID}, eid, sn, rt) = true$

$\mathsf{VerifyTally}(CRS, PK_{eid}, VK_{eid}, \{\mathcal{CT}'_{ID_i}\}_{i=1}^N, M_{sum}, \nu)$:

$\mathcal{CT}'_{sum} = \mathcal{CT}'_{ID_1} \circ \cdots \circ \mathcal{CT}'_{ID_N}$
$assert\ \mathsf{SAVER.Verify\_Dec}(CRS, PK_{eid}, VK_{eid}, M_{sum}, \nu, \mathcal{CT}'_{sum}) = true$

Algorithm 4 Voting system nodes

$\mathsf{PostVote}(CRS, PK_{eid}, rt, ballot)$:

parse $ballot = \{eid, sn, \pi, \mathcal{CT}\}$
$assert\ sn \notin BlockChain$
$assert\ \mathsf{SAVER.Verify\_Enc}(CRS, \pi, \mathcal{CT}, eid, sn, rt) = true$
$\pi', \mathcal{CT}' \leftarrow \mathsf{SAVER.Rerandomize}(PK_{eid}, \pi, \mathcal{CT})$
upload $(eid, sn, \pi', \mathcal{CT}')$ on $BlockChain$

every voter who participates in the system runs GenKey to generate his own sk_{ID} and publish his pk_{ID}.

Phase 1: *open election.* If an administrator wants to open an election, she first selects a list of participants for the election by collecting pk_{ID} of each voter. Then she opens an election distinguished as eid, by calling Election.

Phase 2: *cast vote.* After the election eid is initiated, a voter can run Vote to cast a vote, by sending the transaction *ballot* to the $BlockChain$ network. The $BlockChain$ node runs PostVote to verify the proof, rerandomize the ballot, and post the rerandomized ballot on the $BlockChain$ (the posting can be realized as mining of the block). Then, the voter runs VerifyVote with taking the posted ballot of sn as an input, to ensure the individual verifiability.

Phase 3: *tally results.* When the election is over, the administrator runs Tally with collecting posted ballots as inputs, to publish the result of the election eid.

Algorithm 5 Voting system administrator

relation$(m_1, \ldots, m_n, e, sn, rt;\ \boldsymbol{path}, sk_{ID})$:

$pk_{ID} \leftarrow H(sk_{ID})$
$rt \leftarrow MerkleTree(\boldsymbol{path}, pk_{ID})$
$sn \leftarrow H(e \parallel sk_{ID})$
$assert\ m_i \in \{0,1\}$ for $i = 1$ to n
$assert\ \sum_{i=1}^{n} m_i = 1$

InitSystem(relation) :

$CRS \leftarrow$ SAVER.Setup(relation)
upload CRS on $BlockChain$

Election$(CRS, 1^\kappa, \{pk_{ID_i}\}_{i=1}^N)$:

$\boldsymbol{pklist} \leftarrow \{pk_{ID_i}\}_{i=1}^N$ for total N voters
$rt \leftarrow GetMerkleRoot(\boldsymbol{pklist})$
$eid \xleftarrow{\$} \{0,1\}^\kappa$
$SK_{eid}, PK_{eid}, VK_{eid} \leftarrow$ SAVER.KeyGen(CRS)
return SK_{eid} to $admin$
upload $\boldsymbol{pklist}, PK_{eid}, VK_{eid}, eid, rt$ on $BlockChain$

Tally$(CRS, SK_{eid}, VK_{eid}, \{\mathcal{CT}'_i\}_{i=1}^N)$:

$\mathcal{CT}'_{sum} = \mathcal{CT}'_1 \circ \cdots \circ \mathcal{CT}'_N$
$M_{sum}, \nu \leftarrow$ SAVER.Dec$(CRS, SK_{eid}, VK_{eid}, \mathcal{CT}'_{sum})$
publish (M_{sum}, ν)

Then all the observers can run VerifyTally to ensure the universal verifiability of the result.

7.1 Midterm Audit

In the proposed Vote-SAVER, the administrator can decrypt the ballots and audit the ongoing election results. In certain circumstances, it may even be necessary to prevent such midterm audits. This problem occurs because there is a single administrator who fully holds the decryption key ρ. It can be prohibited by introducing multi-administrators. Unless all administrators collude, auditing the ongoing result is not possible. For the ciphertext for which all administrators provide the decryption information or ν in algorithm 2, the decryption is applicable.

Assume that there are c administrators. Each administrator AD_i chooses ρ_i randomly at KeyGen. And then each AD_i publishes VK_i which is based on ρ_i instead of ρ. Then VK becomes $\prod_{i=1}^{c} VK_i$. At Dec, each AD_i publishes $\nu_i = (\prod_{j=p}^{q} c_{i,0})^{\rho_i}$. By combining ν_i, everyone computes $\nu = \prod_{i=1}^{c} \nu_i$. Using ν, the plaintext is decrypted from the summed ciphertext.

8 Experiment

We implement the proposed SAVER, with respect to the voting relation which is described in section 7. In the relation, Ajtai hash function is adopted as a hash function [Ajt96, KZM+15a] and the tree height is 16. The experiment results are measured on the Ubuntu 18.04 machine with Intel-i5 (3.4GHz) quad-cores and 24GB memory. For the zk-SNARK, we utilized the libsnark [SL14] library.

Table 1: Execution times and parameter sizes in the Vote-SAVER. *ref.* $|M|$ = *message size,* $|m| = 4$ *bits,* $\Pi_{\mathsf{snark}} = [Gro16]$

time	$\|M\|$ (*bits*)				*size*	$\|M\|$ (*bits*)			
	256	512	1024	2048		256	512	1024	2048
Setup	$2.67s$	$2.67s$	$2.69s$	$2.72s$	CRS	$16MB$	$16MB$	$16MB$	$16MB$
KeyGen	$0.01s$	$0.02s$	$0.04s$	$0.09s$	SK	$32B$			
Enc (*sep*)	$1.6ms$	$2.4ms$	$7.4ms$	$8.8ms$	PK	$1246B$	$2321B$	$4465B$	$8753B$
Π_{snark}.Prove	$0.73s$	$0.73s$	$0.73s$	$0.74s$	VK	$1126B$	$2184B$	$4296B$	$8520B$
Verify_Enc	$8.2ms$	$12.7ms$	$21.7ms$	$39.8ms$	$\mathcal{CT}$	$477B$	$749B$	$1293B$	$2381B$
Dec	$37.7ms$	$75.2ms$	$149.7ms$	$300.4ms$	π	$128B$			
Verify_Dec	$14.8ms$	$28.3ms$	$55.5s$	$110.1ms$	ν	$32B$			
Rerandomize	$0.02ms$	$0.03ms$	$0.04ms$	$0.06ms$					

Table 1 shows the execution time for each algorithm, and size for the parameters. We vary the message size from 256 bits to 2048 bits, where the message is a ballot for list of candidates. For instance, an integer vote in which 4 bytes data is used for each candidate can represent 8 candidates. We fix the message block size as $|m| = 32bits$ for all message spaces. For example, 256-bit M consists of 8 blocks of messages. The block size determines the ciphertext size and decryption time. A larger block size can yield less number of total blocks, which leads to less number of ciphertext blocks to decrease the ciphertext size. However, as a trade-off, it increases the decryption time due to the increased computation of discrete log search. Since we fix the block size, the decryption time is strictly linear to the message size which determines the number of message blocks.

The Enc in SAVER consists of a normal encryption and Π_{snark}.Prove for the voting relation (i.e. membership tests and range checks); Enc (*sep*) is a separated time for the normal encryption. The zk-SNARK proving time takes 0.74s, which is dominant in the total encryption time, while the normal encryption takes less than 8ms for $|M| = 2048bits$. In the SAVER, the number of elements for PK, VK and $\mathcal{CT}$ is determined by the number of message blocks. Therefore it is shown in the result that $PK, VK, \mathcal{CT}$ size increases along with the message size. For the fixed relation, CRS size remains as 16MB for all message sizes, which is practical to be stored in the portable devices.

9 Conclusion

This paper proposes SAVER: Snark-friendly, Additively-homomorphic, and Verifiable Encryption and decryption with Rerandomization, which is universal verifiable encryption achieved from connecting zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARK) and verifiable encryption. The proposed SAVER satisfies many useful functionalities. It is *snark-friendly*, to be compatible with the pairing-based zk-SNARKs. It is *additively-homomorphic*, so that the ciphertexts can be merged additively. It is a *verifiable encryption*, which can prove arbitrary properties of the message. It is a *verifiable decryption*, which can prove validity of the decryption. It provides *rerandomization*, where the ciphertext can be rerandomized as a new encryption. The security of the proposed SAVER is formally proved.

This paper also represents a Vote-SAVER achieved by applying the proposed SAVER, which satisfies the ideal properties required by voting systems. The Vote-SAVER satisfies receipt-freeness, eligibility verifiability, individual verifiability, and universal verifiability altogether, which resolves long-lasting problem on the conflict between receipt-freeness and individual verifiability. The experiment results show that the proposed SAVER yields the encryption time of 8.8ms excluding proving time and the CRS size of 16MB for 2048-bit message, which is very practical compared to the encryption-in-the-circuit approach.

References

[Adi08] Ben Adida. Helios: Web-based open-audit voting. In *USENIX security symposium*, volume 17, pages 335–348, 2008.

[AHL+12] Nuttapong Attrapadung, Javier Herranz, Fabien Laguillaumie, Benoît Libert, Elie De Panafieu, and Carla Ràfols. Attribute-based encryption schemes with constant-size ciphertexts. *Theoretical computer science*, 422:15–38, 2012.

[Ajt96] Miklós Ajtai. Generating hard instances of lattice problems (extended abstract). In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, pages 99–108, 1996.

[AM16] Syed Taha Ali and Judy Murray. An overview of end-to-end verifiable voting systems. *Real-world electronic voting: Design, analysis and deployment*, pages 171–218, 2016.

[Ate04] Giuseppe Ateniese. Verifiable encryption of digital signatures and applications. *ACM Transactions on Information and System Security (TISSEC)*, 7(1):1–20, 2004.

[BBG05] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. In Ronald Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, volume 3494 of *Lecture Notes in Computer Science*, pages 440–456. Springer, 2005.

[BCCT12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 326–349. ACM, 2012.

[BCG+14] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014*, pages 459–474, 2014.

[BG18] Sean Bowe and Ariel Gabizon. Making groth's zk-snark simulation extractable in the random oracle model. Cryptology ePrint Archive, Report 2018/187, 2018. https://eprint.iacr.org/2018/187.

[BSCTV17] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Scalable zero knowledge via cycles of elliptic curves. *Algorithmica*, 79(4):1102–1160, 2017.

[BT94] Josh Benaloh and Dwight Tuinstra. Receipt-free secret-ballot elections. In *STOC*, volume 94, pages 544–553, 1994.

[CD00] Jan Camenisch and Ivan Damgård. Verifiable encryption, group encryption, and their applications to separable group signatures and signature sharing schemes. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 331–345. Springer, 2000.

[CFQ19] Matteo Campanelli, Dario Fiore, and Anaïs Querol. Legosnark: Modular design and composition of succinct zero-knowledge proofs. Cryptology ePrint Archive, Report 2019/142, 2019. https://eprint.iacr.org/2019/142.

[CGS97] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally efficient multi-authority election scheme. *European transactions on Telecommunications*, 8(5):481–490, 1997.

[CL18] Véronique Cortier and Joseph Lallemand. Voting: You can't have privacy without individual verifiability. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 53–66. ACM, 2018.

[CRST15] Chris Culnane, Peter YA Ryan, Steve Schneider, and Vanessa Teague. vvote: a verifiable voting system. *ACM Transactions on Information and System Security (TISSEC)*, 18(1):3, 2015.

[CS03] Jan Camenisch and Victor Shoup. Practical verifiable encryption and decryption of discrete logarithms. In *Annual International Cryptology Conference*, pages 126–144. Springer, 2003.

[FFG+16] Dario Fiore, Cédric Fournet, Esha Ghosh, Markulf Kohlweiss, Olga Ohrimenko, and Bryan Parno. Hash first, argue later: Adaptive verifiable computations on outsourced data. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1304–1316. ACM, 2016.

[Gab19] Ariel Gabizon. On the efficiency of pairing-based proofs under the d-pke. Cryptology ePrint Archive, Report 2019/148, 2019. https://eprint.iacr.org/2019/148.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct nizks without pcps. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 626–645. Springer, 2013.

[GM17] Jens Groth and Mary Maller. Snarky signatures: Minimal signatures of knowledge from Simulation-Extractable SNARKs. In *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part II*, pages 581–612, 2017.

[GPS08] Steven D. Galbraith, Kenneth G. Paterson, and Nigel P. Smart. Pairings for cryptographers. *Discrete Applied Mathematics*, 156(16):3113–3121, 2008.

[Gro16] Jens Groth. On the size of Pairing-Based non-interactive arguments. In *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*, pages 305–326, 2016.

[HS00] Martin Hirt and Kazue Sako. Efficient receipt-free voting based on homomorphic encryption. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 539–556. Springer, 2000.

[IKSA03] Subariah Ibrahim, Maznah Kamat, Mazleena Salleh, and Shah Rizan Abdul Aziz. Secure e-voting with blind signature. In *4th National Conference of Telecommunication Technology, 2003. NCTT 2003 Proceedings.*, pages 193–197. IEEE, 2003.

[JMP13] Hugo Jonker, Sjouke Mauw, and Jun Pang. Privacy and verifiability in voting systems: Methods, developments and trends. *Computer Science Review*, 10:1–30, 2013.

[KLLO18] Jihye Kim, Seunghwa Lee, Jiwon Lee, and Hyunok Oh. Scalable wildcarded identity-based encryption. In *European Symposium on Research in Computer Security*, pages 269–287. Springer, 2018.

[KLO19] Jihye Kim, Jiwon Lee, and Hyunok Oh. Qap-based simulation-extractable snark with a single verification. Cryptology ePrint Archive, Report 2019/586, 2019. https://eprint.iacr.org/2019/586.

[KZM+15a] Ahmed Kosba, Zhichao Zhao, Andrew Miller, Yi Qian, H Chan, Charalampos PAPAMAN-THOU, Rafael Pass, SHELAT ABHI, and EC SHI. c: A framework for building composable zero-knowledge proofs. Technical report, Cryptology ePrint Archive, Report 2015/1093, 2015. http://eprint. iacr. org . . . , 2015.

[KZM+15b] Ahmed E Kosba, Zhichao Zhao, Andrew Miller, Yi Qian, T-H Hubert Chan, Charalampos Papamanthou, Rafael Pass, Abhi Shelat, and Elaine Shi. How to use snarks in universally composable protocols. *IACR Cryptology ePrint Archive*, 2015:1093, 2015.

[Lip19] Helger Lipmaa. Simulation-extractable snarks revisited. 2019.

[LN17] Vadim Lyubashevsky and Gregory Neven. One-shot verifiable encryption from lattices. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 293–323. Springer, 2017.

[LW17] Yi Liu and Qi Wang. An e-voting protocol based on blockchain. *IACR Cryptology ePrint Archive*, 2017:1043, 2017.

[MN06] Tal Moran and Moni Naor. Receipt-free universally-verifiable voting with everlasting privacy. In *Annual International Cryptology Conference*, pages 373–392. Springer, 2006.

[Oka97] Tatsuaki Okamoto. Receipt-free electronic voting schemes for large scale elections. In *International Workshop on Security Protocols*, pages 25–35. Springer, 1997.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 223–238. Springer, 1999.

[PHGR13] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *2013 IEEE Symposium on Security and Privacy*, pages 238–252. IEEE, 2013.

[PR07] Manoj Prabhakaran and Mike Rosulek. Rerandomizable rcca encryption. In *Annual International Cryptology Conference*, pages 517–534. Springer, 2007.

[RBH+09] Peter YA Ryan, David Bismark, James Heather, Steve Schneider, and Zhe Xia. Prêt à voter: a voter-verifiable voting system. *IEEE transactions on information forensics and security*, 4(4):662–673, 2009.

[RS17] Ronald L Rivest and Philip B Stark. When is an election verifiable? *IEEE Security & Privacy*, (3):48–50, 2017.

[Sak11] Kazue Sako. *Verifiable Encryption*, pages 1356–1357. Springer US, Boston, MA, 2011.

[SL14] SCIPR-Lab. libsnark. https://github.com/scipr-lab/libsnark, 2014.

[Smy18] Ben Smyth. A foundation for secret, verifiable elections. *IACR Cryptology ePrint Archive*, 2018:225, 2018.

[YAS+12] Shota Yamada, Nuttapong Attrapadung, Bagus Santoso, Jacob CN Schuldt, Goichiro Hanaoka, and Noboru Kunihiro. Verifiable predicate encryption and applications to cca security and anonymous predicate authentication. In *International Workshop on Public Key Cryptography*, pages 243–261. Springer, 2012.