Tangram: Encryption-friendly SNARK framework under Pedersen committed engines

Gweonho Jeong*, Myeongkyun Moon*, Geonho Yoon*, Hyunok Oh*, and Jihye Kim†

***Abstract*—SNARKs are frequently used to prove encryption, yet the circuit size often becomes large due to the intricate operations inherent in encryption. It entails considerable computational overhead for a prover and can also lead to an increase in the size of the public parameters (e.g., evaluation key). We propose an encryption-friendly SNARK framework, Tangram, which allows anyone to construct a system by using their desired encryption and proof system. Our approach revises existing encryption schemes to produce Pedersen-like ciphertext, including identity-based, hierarchical identity-based, and attribute-based encryption. Afterward, to prove the knowledge of the encryption, we utilize a modular manner of commit-and-prove SNARKs, which uses commitment as a 'bridge'. With our framework, one can prove encryption significantly faster than proving the whole encryption within the circuit. We implement various Tangram gadgets and evaluate their performance. Our results show 12x - 3500x times better performance than encryption-in-the-circuit.**

***Index Terms*—zk-SNARKs, Pedersen commitment, Encryptions, CP-SNARK**

I. Introduction

ECRYPTION is a fundamental cryptographic primitive extensively used across diverse applications to ensure data privacy and confidentiality. As the importance of personal privacy gains prominence, numerous applications, including blockchain, digital credentials, and the IoT, rely heavily on encryption to safeguard transactional data and sensitive information. In these contexts, there is a need to combine encryption with proof systems to verify encrypted data meets certain publicly defined properties without compromising the data's confidentiality. For example, Central Bank Digital Currency (CBDC) systems and privacy-preserving blockchain platforms employ encryption to regulate and audit user activities effectively [1, 2, 3, 4].

Utilizing zk-SNARKs [5, 6, 7, 8, 9], which allow for proving the properties of encrypted data without revealing the underlying information, addresses the growing need for privacy-preserving technologies. While zk-SNARKs offer efficient verification and compact proof sizes, they present significant challenges for the prover, particularly when dealing with encryption schemes involving group operations. These challenges include high computational overhead for the prover and the requirement for large public parameters, which restricts the use of encryption to only simplified and basic forms. Moreover, integrating more sophisticated encryption schemes combined with access control [10, 11, 12] which often includes pairing operations, becomes almost impossible due to the extreme overhead incurred in the SNARK circuit. To effectively integrate zk-SNARKs into practical privacy-preserving services, it is essential to develop methods for efficiently proving these complex encryption schemes.

There exist studies [13, 14, 15, 16, 17] to address the issues mentioned above. Typically, these schemes perform large computations outside the circuit and then connect computed results with identical values. One notable work is Saver [17], which addresses the inefficiency of proving encryption within the circuit by designing a SNARK-friendly verifiable encryption scheme based on a specific proof system [5]. Saver generates ciphertexts outside the circuit and integrates them directly into the proof system, thus improving proof generation time for encryption. However, Saver is limited to ElGamal encryption and cannot be used with encryption schemes that support additional properties, such as granting access to encrypted data only to individuals with specific identifying information or users who satisfy certain attributes or conditions. Moreover, the encryption key is circuit-dependent, as it is derived from a common reference string.

Another notable example is LegoSNARK [16], which presents the general commit-and-prove SNARK framework. This framework facilitates the connection of proof systems using commitments as a 'bridge'. LegoSNARK also introduces a limited version of commit-and-prove SNARK, called *commit-carrying* SNARK ($\mathsf{SNARK}_{\mathsf{cc}}$), in which a generated commitment is proof-dependent as the commitment scheme is bound to the relation. LegoSNARK can efficiently prove the circuit of Pedersen computation using $\mathsf{SNARK}_{\mathsf{cc}}$ and a linkable proof system. In this structure, the linkable proof system proves the equality of the messages in two different Pedersen commitments. However, since advanced encryption schemes such as [10, 11, 12] output ciphertexts that differ from the format of Pedersen commitments, LegoSNARK cannot be used to efficiently prove these ciphertexts.

To address these issues while mitigating their limitations, we investigate and revise existing encryption schemes to output ciphertexts in a compatible group format, utilizing the benefits of zk-SNARK proof systems and linkable proof systems. Our goal is to ensure efficient proof generation time when proving encryption, ranging from simple to advanced, using

* Gweonho Jeong is with Hanyang University, Seoul, Korea (email: kwonhojeong@hanyang.ac.kr).
* Myeongkyun Moon is with Hanyang University, Seoul, Korea (email: civilization@hanyang.ac.kr).
* Geonho Yoon is with Hanyang University, Seoul, Korea (email: geonho@hanyang.ac.kr).
* Hyunok Oh is with Hanyang University, Seoul, Korea (email: hoh@hanyang.ac.kr).
† Jihye Kim is with Kookmin University, Seoul, Korea (email: jihyek@kookmin.ac.kr).

zk-SNARKs under Pedersen committed engines.[1]

A. Technical Overview

In this section, we describe a high-level technical overview of our scheme. Consider an arbitrary relation $\mathcal{R}$ constructed as follows:

$$\mathcal{R} := \mathcal{R}_{\mathsf{Enc}} \wedge \mathcal{R}_{\mathsf{Prop}} = \{\mathsf{ct} \leftarrow \mathsf{Enc}(m) \wedge \mathsf{Prop}(m)\}$$

where Prop denotes an additional property. Our rough technique to gain fast prover time for encryption consists of two parts: 1) take out the encryption part $\mathcal{R}_{\mathsf{Enc}}$ of the entire relation $\mathcal{R}$ and 2) show *connectivity* between the outer encryption and the remaining relation $\mathcal{R}_{\mathsf{Prop}}$ by leveraging a linking proof system.

Our main idea starts with defining a variant of encryption, termed as *commit-carrying encryption*, which yields a ciphertext resembling the form of commitment. In particular, we aim for encryption outputting a Pedersen-like ciphertext ($\mathsf{ct} \approx \mathsf{Ped.Com}(m)$). Fortunately, some encryption schemes, like additively-homomorphic ElGamal encryption, fulfill this property. However, many schemes fail to meet this condition. For example, some schemes produce ciphertexts comprising different group elements, such as ($\mathbb{G}_1$ and $\mathbb{G}_2$). Moreover, even if an encryption scheme produces a Pedersen-like ciphertext, it often has a large domain size, like $\mathbb{G}_T$, posing challenges for directly plugging into the proof system.

To solve the problem, our approach is to revise the ciphertext format to Pedersen-like format while retaining the properties of the original encryption scheme. For example, in identity-based encryption (IBE) [18], the ciphertext for a message m is constructed as

$$\mathsf{ct} = (\mathsf{ct}_0, \mathsf{ct}_1, \mathsf{ct}_2) = \left(e(\square, \square)^t \cdot m, \square^t, \square^t\right)$$

where t is random, e is a pairing, and $\square$ represents arbitrary group elements. While ct_1 and ct_2 can be viewed as commitments to a zero message, ct_0 is constructed differently and does not match a Pedersen-like commitment. However we observe it can be revised by domain adjustment and restructuring to form an additively-homomorphic ElGamal ciphertext. Then, if we show that the revised scheme is secure under cryptographic assumption, the modified encryption scheme can be regarded as a commit-carrying IBE encryption. Based on this approach, we propose instantiations of various commit-carrying encryption schemes, including advanced encryption ones like hierarchical-identity-based encryption (HIBE) [11] and attribute-based encryption (ABE) [12].

By leveraging commit-carrying encryption, we can generate a ciphertext regarded as a Pedersen commitment for a message m. In parallel, with the identical message m, we can prove the remaining relation $\mathcal{R}_{\mathsf{Prop}}$ using commit-carrying SNARK introduced by LegoSNARK. When proving the knowledge through it, the prover $\mathcal{P}$ requires a witness $\boldsymbol{w} = (u, \omega)$ and an instance $\boldsymbol{x}$ as input, and then generates a proof π along with a commitment cm where u is a committed witness. If $\mathsf{cm} \leftarrow \mathsf{Com}(\mathsf{ck}, u) \approx \mathsf{Ped.Com}(\mathsf{ck}, u)$, the commit-carrying SNARK is viewed as the proof system runs under Pedersen engine. Through this, we can achieve the separation of the $\mathcal{R}_{\mathsf{Enc}}$ that we intend in the first part.

Several types of zk-SNARKs can meet the requirement for a Pedersen committed engine. For example, $\mathsf{Gro16}$ [5] uses a common key about an arbitrary relation $\mathcal{R}$ to create a proof through linear encoding. It can transform into a commit-carrying SNARK, illustrated in LegoSNARK [16]. Meanwhile, Plonk [6] uses a polynomial commitment scheme (PCS) to encode the witness-encoded polynomial $\boldsymbol{w}(X)$ for a proof, resulting in $\mathsf{cm} = \mathsf{PCS.Com}(w(X))$. The polynomial can be split into two components: $\boldsymbol{w}_{\mathsf{cm}}(X)$ and $\boldsymbol{w}_{\mathsf{aux}}(X)$, where $\boldsymbol{w}_{\mathsf{cm}}(X)$ is to represent the committed part (i.e. u) of the witness, while $\boldsymbol{w}_{\mathsf{aux}}(X)$ is the non-committed part. It means that $\boldsymbol{w}(X)$ can be expressed as the composition of $\boldsymbol{w}_{\mathsf{cm}}(X)$ and $\boldsymbol{w}_{\mathsf{aux}}(X)$. If Plonk utilizes KZG commitment [19] and a blind factor is added to the $\boldsymbol{w}_{\mathsf{cm}}(X)$ side to output a commitment to u, it can be seen as the commit-carrying SNARK.

When using the two components mentioned above, we can obtain two commitments to the same message: one is generated outside the circuit from the commit-carrying encryption, and the other is internally created from the commit-carrying SNARK. All that remains is assigning *connectivity* between these two components. This connection is facilitated by a proof system, which can prove they used the same message in their Pedersen commitments to different keys. We name the proof system Linker. There are many systems to prove the relation, such as NIZK for linear subspace [20], and compressed Σ-protocol [21, 22].

B. Our Contributions

Design of commit-carrying encryption. We define commit-carrying encryption (cc-Enc) in which an encryptor encrypts a message while committing to a value within the ciphertext. We instantiate specific commit-carrying encryption schemes that produce Pedersen commitment-like ciphertexts. These can easily connect to different combinations of SNARKs, making them versatile.

Encryption-friendly SNARK framework. We propose the first encryption-friendly SNARK framework under Pedersen committed engines with efficient prover time, coined $\mathsf{Tangram}$. There are three main recipes required to build the $\mathsf{Tangram}$ framework: *commit-carrying encryption* ($\mathsf{cc\text{-}Enc}$), *commit-carrying SNARK* ($\mathsf{SNARK}_{\mathsf{cc}}$), and *commitment-linking protocol* (Linker), all based on the Pedersen committed engine. This modularity allows for bespoke $\mathsf{Tangram}$ constructions. As an illustration, by combining these components, one can create a variety of frameworks, such as a trusted setup framework with a small proof size for identity-based encryption or a framework with universal SNARK properties for attributed-based encryption.

[1]Pedersen committed engines mean that the witness of the proof system is committed using Pedersen commitments. Since commit-and-prove SNARKs [16] leverage Pedersen commitments to enable the combination of different proof systems, our proposed framework, as an extension of commit-and-prove SNARKs, focuses on Pedersen-committed engines to enhance modularity and interoperability.

Implementation and evaluation. To evaluate the performance of the Tangram framework, we implement several gadgets. Using these, we construct different versions of the Tangram framework and evaluate them against standard methods like encryption-in-the-circuit on Groth16 [5] and PLONK [6]. Our experiments show that Tangram improves at least 12x in proving time depending on the chosen commit-carrying encryption. In the context of key size, there is a significant improvement. For instance, in the case of ElGamal using Groth16, public parameters (e.g., evaluation key, verification key) are approximately 60x larger, measuring around 1.5 MB, in contrast to about 24.8 KB for applying our framework. Although a drawback of Tangram is larger proof/ciphertext size and slower decryption time, yet it shows a reasonable level of speed/size for use: decryption time takes $\approx$ 17ms (vs 0.5 ms), proof size is $\approx$ 0.3KB (vs 288B), and ciphertext size is $\approx$ 3.3KB (vs 64B).

C. Related works

After zero-knowledge proofs were introduced in [23], there has been significant progress in efficient ZKP protocols and systems. We can classify ZKP systems based on underlying techniques; there exist systems based on bilinear maps [24, 5, 25, 9, 6, 8, 26], interactive proofs [27, 28], discrete logarithm [29, 30, 7, 31], interactive oracle proofs (IOP) [32, 33, 34], MPC-in-the-head [35, 36] and lattices [37, 38].

Commit-and-prove SNARK. Among the diverse research areas mentioned above, the idea of combining two distinct NIZKs has been explored by various research streams with the aim of mitigating the overhead faced by the prover due to extensive computations in a circuit, as seen in prior work [13, 14, 15, 16]. In the case of LegoSNARK [16], the paper proposes a general framework for constructing $\mathsf{SNARK}_{\mathsf{cp}}$ in a modular manner. Numerous $\mathsf{SNARK}_{\mathsf{cp}}$s exist, some of which can be compiled into $\mathsf{SNARK}_{\mathsf{cp}}$ [24, 13, 17], and those inherently $\mathsf{SNARK}_{\mathsf{cp}}$ [39, 13, 40, 41, 27, 42, 7]. It describes a compiler that transforms $\mathsf{SNARK}_{\mathsf{cc}}$ into $\mathsf{SNARK}_{\mathsf{cp}}$ using the NIZK scheme to prove that two distinct Pedersen-like commitments open to the same vector. Another work, Lunar [43], builds $\mathsf{SNARK}_{\mathsf{cp}}$ with a universal and updatable SRS and introduces proof systems linking committed inputs to the polynomial commitments employed in Algebraic Holographic Proofs (AHP)-based arguments. The recent work, Eclipse [44], proposes a compiler that turns an AHP-based proof system into $\mathsf{SNARK}_{\mathsf{cp}}$ and instantiates $\mathsf{SNARK}_{\mathsf{cp}}$s for Plonk [6], Sonic [9], and Marlin [8] with logarithmic proof size using compressed Σ-protocol. However, we contend that it is feasible to extend the ideas presented in previous papers beyond commit schemes to also encompass encryption.

Verifiable encryption based on SNARK. A work closely related to ours is SAVER [17]. It constructs a verifiable encryption scheme tailored to use in voting schemes as it is additively homomorphic and supports rerandomization. Concretely, the scheme of SAVER utilizes that Gro16 is constructed as group linear encoding in which it is possible to extend the statement as an ElGamal ciphertext. Nonetheless, this construction is tightly coupled with the specific proof system and encryption method, making it difficult to be flexibly modified to suit the preferences of applications. Nick et al. [45] proposed a construction that can encrypt a discrete logarithm in an elliptic curve group using a specialized PRF called Purify. The scheme offers the capability to encrypt an ECDSA private key without requiring any trusted setup assumption, owing to the usage of Bulletproofs. However, the encryption must be performed using public key encryption like ElGamal. Notably, the proof system and encryption approach are tightly coupled, making modifying the scheme according to the users' requirements challenging.

II. Preliminaries

A. Notations

We use $\boldsymbol{a}$ or $\{a_i\}$ for the list of elements. We denote by λ a security parameter and by $\epsilon(\cdot)$ a negligible function. Let $\mathbb{F}$ denote a finite field and $\mathbb{G}$ denote a group. A bilinear group generator $\mathcal{BG}$ takes a security parameter as input in unary and returns a bilinear group $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ consisting of cyclic groups $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ of prime order p and a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. Given the security parameter 1^λ, a relation generator $\mathcal{RG}$ returns a polynomial time decidable relation $\mathcal{R} \leftarrow \mathcal{RG}(1^\lambda)$. For $(\boldsymbol{x}, \boldsymbol{w}) \in \mathcal{R}$ we say $\boldsymbol{w}$ is a witness to the statement $\boldsymbol{x}$ being in the relation.

B. Public key encryption

A public key encryption scheme consists of a tuple of three algorithm $\mathbf{\Pi}_{\mathsf{PKE}}$ = (KeyGen, Enc, Dec). Let $\mathcal{D}_m$ be a message domain and $\mathcal{D}_r$ be a domain from which randomness is sampled.

- $\mathsf{KeyGen}(1^\lambda)$: takes a security parameter 1^λ as input, and KeyGen outputs a key pair $(\mathsf{pk}, \mathsf{sk})$
- $\mathsf{Enc}(\mathsf{pk}, m)$: takes a public key pk and a message $m \in \mathcal{D}_m$ as inputs, and Enc outputs a ciphertext ct.
- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct})$: takes a secret key sk and a ciphertext ct as inputs, and Dec outputs the plaintext m.

Definition 1. PKE provides indistinguishable (IND-CPA) security if for any PPT adversary the following probability is negligible,

$$\left\| \Pr\left[\begin{matrix} (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda); (m_0, m_1) \leftarrow \mathcal{A}(\mathsf{pk}) \\ b \xleftarrow{\$} \{0,1\}; \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b); b' \leftarrow \mathcal{A}(\mathsf{ct}) \end{matrix} : b = b' \right] - \frac{1}{2} \right\|$$

C. Pedersen vector commitment

Pedersen vector commitment for vector $\boldsymbol{w}$ of size n can be expressed succinctly with the following algorithms:

- $\mathsf{Ped.Setup}(1^\lambda)$: chooses $g \xleftarrow{\$} \mathbb{G}$, $\boldsymbol{h} \xleftarrow{\$} \mathbb{G}^n$ from a domain $\mathcal{D}$. It outputs a commitment key $\mathsf{ck} := (g, \boldsymbol{h})$.
- $\mathsf{Ped.Com}(\mathsf{ck}, \boldsymbol{w})$: chooses an opening $o \xleftarrow{\$} \mathbb{Z}_q$ and returns $(\mathsf{cm}, o) := ((o, \boldsymbol{w})^\top \cdot \mathsf{ck}, o)$.
- $\mathsf{Ped.VerCom}(\mathsf{ck}, \mathsf{cm}, \boldsymbol{w}, o)$: returns true if cm = $(o, \boldsymbol{w}^\top)$ $\cdot$ ck. Otherwise, false.

The Pedersen vector commitment is perfectly hiding and computationally binding if the discrete logarithm assumption holds.

D. *Succinct Non-interactive arguments of knowledge*

Definition 2. A succinct non-interactive arguments of knowledge (SNARK) for $\mathcal{R}$ is a tuple of algorithms $\mathbf{\Pi}_{\mathsf{snark}}$ =(Setup, Prove, Verify) working as follows:

- $\mathsf{crs} := (\mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{Setup}(\mathcal{R})$: takes a relation $\mathcal{R} \leftarrow \mathcal{RG}(1^\lambda)$ as input and returns a common reference string crs consisting of an evaluation key ek and a verification key vk.
- $\pi \leftarrow \mathsf{Prove}(\mathsf{ek}, \boldsymbol{x}; \boldsymbol{w})$: takes an evaluation key ek, a statement $\boldsymbol{x}$, and a witness $\boldsymbol{w}$ as inputs, and returns a proof π.
- $\mathsf{true}/\mathsf{false} \leftarrow \mathsf{Verify}(\mathsf{vk}, \boldsymbol{x}, \pi)$: takes a verification key vk, a statement $\boldsymbol{x}$, and a proof π as inputs and returns false $(reject)$ or true $(accept)$.

It satisfies completeness, knowledge soundness, and succinctness

A SNARK may also satisfy *zero-knowledge*. 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). We refer to it as a zk-SNARK in this scenario.

E. *Commit-and-Prove SNARK*

The commit-and-prove SNARK ($\mathsf{SNARK}_{\mathsf{cp}}$) is a variant of a SNARK, which combines the proof systems using their commitments as a 'bridge'. LegoSNARK [16] defines a framework for $\mathsf{SNARK}_{\mathsf{cp}}$ that provides general composition tools for modularly building new $\mathsf{SNARK}_{\mathsf{cp}}$ from proof gadgets. To explain further, a $\mathsf{SNARK}_{\mathsf{cp}}$ for the relation $\mathcal{R}(\boldsymbol{x}; \boldsymbol{w})$ is a type of SNARK that can prove knowledge of $\boldsymbol{w} = (u, \omega, o)$ for a given commitment cm, where $\mathsf{cm} = \mathsf{Com}(u; o)$ and $\mathcal{R}(\boldsymbol{x}; u, \omega)$ is true. We can regard ω as a non-committed part of the witness $\boldsymbol{w}$. In $\mathsf{SNARK}_{\mathsf{cp}}$, the verifier's inputs consist of the proof, as well as the values of x and cm. In short, $\mathsf{SNARK}_{\mathsf{cp}}$ is a tuple of algorithms $\mathbf{\Pi}_{\mathsf{cp}}$ as defined below.

- $\mathsf{crs} \leftarrow \mathsf{KeyGen}(\mathsf{ck}, \mathcal{R})$: takes a commitment key ck, a relation $\mathcal{R}$ as inputs, and outputs a common reference string $\mathsf{crs} := (\mathsf{ek}, \mathsf{vk})$.
- $\pi \leftarrow \mathsf{Prove}(\mathsf{ek}, \boldsymbol{x}; (u_i)_{i\in[l]}, (o_i)_{i\in[l]}, \omega)$: takes an evaluation key ek, a statement $\boldsymbol{x} = (x, (\mathsf{cm}_i)_{i\in[l]})$, messages u_i, randomnesses o_i, non-committed witnesses w, and outputs a proof π.
- $0/1 \leftarrow \mathsf{Verify}(\mathsf{vk}, \boldsymbol{x}, \pi)$: takes a verification key vk, a statement $\boldsymbol{x} = (x, (\mathsf{cm}_i)_{i\in[l]})$, and a proof π. It rejects or accepts the proof.

1) Commit-carrying SNARK: There exists a variant of $\mathsf{SNARK}_{\mathsf{cp}}$, referred to as a commit-carrying SNARK ($\mathsf{SNARK}_{\mathsf{cc}}$), which is a SNARK whose proof includes a commitment to a portion of the witnesses. It also satisfies completeness, succinctness, knowledge soundness, and binding. $\mathsf{SNARK}_{\mathsf{cc}}$ consists of a set of algorithms $\mathbf{\Pi}_{\mathsf{cc}}$ as follows.

- $(\mathsf{ck}, \mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(\mathcal{R})$: takes a relation $\mathcal{R}$ as inputs, and outputs a common reference string which includes a commitment key ck, an evaluation key ek, and a verification key vk.
- $(\mathsf{cm}, \pi; o) \leftarrow \mathsf{Prove}(\mathsf{ek}, \boldsymbol{x}; \boldsymbol{w})$: takes an evaluation key ek, a statement $\boldsymbol{x}$ and a witness $\boldsymbol{w} := (u, \omega)$ such that the relation $\mathcal{R}$ holds on the inputs, and outputs a proof π, a commitment cm and an opening o such that $\mathsf{VerCom}(\mathsf{ck}, \mathsf{cm}, u, o) = \mathsf{true}$.
- $\mathsf{true}/\mathsf{false} \leftarrow \mathsf{Verify}(\mathsf{vk}, \boldsymbol{x}, \mathsf{cm}, \pi)$: takes a verification key vk, a statement $\boldsymbol{x}$, a commitment cm, a proof π as inputs, and outputs true if $(\boldsymbol{x}, \mathsf{cm}, \pi) \in \mathcal{R}$, or false otherwise.
- $\mathsf{true}/\mathsf{false} \leftarrow \mathsf{VerCom}(\mathsf{ck}, \mathsf{cm}, u, o)$: takes a commitment key ck, a commitment cm, a message u and an opening o as inputs, and outputs true if the commitment and opening is correct, or false otherwise.

III. Commit-Carrying Encryption

Intuitively, given $\mathbf{\Pi}_{\mathsf{PKE}} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$, if a generated ciphertext ct can be seen as a Pedersen commitment cm, then encryption can be thought of as a commitment scheme where the ciphertext serves as a commitment to the plaintext. Similar definitions exist as committing encryption and research introducing the concept in [46, 17, 47]. We re-formalize the encryption scheme to better suit our construction and then define a notion called *commit-carrying encryption*, which outputs a ciphertext that simultaneously performs the role of a commitment constructed with an encryption-dependent commitment key.

Definition 3. Commit-Carrying Encryption (cc-Enc) A commit-carrying encryption is a tuple of algorithms $\mathbf{\Pi}_{\mathsf{cc\text{-}Enc}}$ = (KeyGen, Enc, Dec, VerEnc) that works as defined below.

- $(\mathsf{pp}, \mathsf{aux}) \leftarrow \mathsf{Setup}(1^\lambda)$: takes the security parameter λ as input, and outputs a public parameter pp and an auxiliary value.
- $(\mathsf{ck}, \mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp}, \mathsf{aux})$: takes the public parameter and auxiliary value as inputs, and outputs a commitment key ck, a public key pk and a secret key sk.
- $(\mathsf{ct}; r) \leftarrow \mathsf{Enc}(\mathsf{pp}, \mathsf{pk}, m)$: takes a public parameter pp, a public key pk, and messages m as inputs, and outputs the ciphertext ct and a randomness r.
- $m \leftarrow \mathsf{Dec}(\mathsf{pp}, \mathsf{sk}, \mathsf{ct})$: takes a public parameter pp, a secret key sk and the ciphertext ct as inputs, and outputs original messages m.
- $\mathsf{true}/\mathsf{false} \leftarrow \mathsf{VerEnc}(\mathsf{ck}, \mathsf{ct}, r, m)$: takes a commitment key ck, a ciphertext ct, a randomness r and a message m, and outputs accepts true, or false otherwise.

In the case of advanced encryption, as opposed to simple public key encryption, additional properties may require the inclusion of an auxiliary input denoted as aux. For instance, in identity-based encryption, elements related to the identity and the master key are needed, while in attribute-based encryption, the access structure is required. Due to the diverse forms of encryption, generalization becomes challenging; hence, we use aux to represent the varying auxiliary input for each algorithm.

Lemma 1. If cc-Enc is perfect correct and IND-CPA secure, then cc-Enc is binding and hiding.

Proof. We prove that $\mathsf{cc\text{-}Enc}$ satisfies the aforementioned security properties.

Binding. It is satisfied by perfect correctness. Consider the scenario in which there exists an adversary that outputs a tuple $(m_0, r_0, m_1, r_1, \mathsf{ct})$ s.t., $m_0 \neq m_1$, $(\mathsf{ct}; r_0) \leftarrow \mathsf{Enc}(\mathsf{pp}, \mathsf{pk}, m_0)$, and $(\mathsf{ct}; r_1) \leftarrow \mathsf{Enc}(\mathsf{pp}, \mathsf{pk}, m_1)$. It breaks binding with non-negligible probability. Given such an efficient adversary $\mathcal{A}$ against the binding game, we construct another adversary $\mathcal{B}$ that uses $\mathcal{A}$ to break perfect correctness. The adversary $\mathcal{B}$ takes $(\mathsf{pk}, \mathsf{sk})$ as inputs, and then sends it to $\mathcal{A}$. When $\mathcal{A}$ outputs $(m_0, r_0, m_1, r_1, \mathsf{ct})$ such that $(\mathsf{ct}, r_0) = \mathsf{Enc}(\mathsf{pp}, \mathsf{pk}, m_0)$, $(\mathsf{ct}, r_1) = \mathsf{Enc}(\mathsf{pp}, \mathsf{pk}, m_1)$ and $m_0 \neq m_1$, $\mathcal{B}$ decrypts the ciphertext as $m = \mathsf{Dec}(\mathsf{pp}, \mathsf{sk}, \mathsf{ct})$ and then outputs as follows.

$$\begin{cases} (\mathsf{ct}, m_0, r_0), & \text{if } m \neq m_0 \\ (\mathsf{ct}, m_1, r_1), & \text{if } m \neq m_1 \end{cases}$$

Since $m_0 \neq m_1$, one of them must occur with certainty. Hence, if the adversary $\mathcal{A}$ can non-negligibly break binding, then $\mathcal{B}$ can also break perfect correctness with the same probability.

Hiding. It is satisfied by IND-CPA. If a PPT distinguisher exists for commitment $(\mathsf{ct}; r) = \mathsf{Com}(\mathsf{ck}, m) \approx \mathsf{PKE.Enc}(\mathsf{pk}, m)$, an IND-CPA distinguisher can be constructed. To break the IND-CPA game, the reduction first receives two messages (m_0, m_1) from the hiding adversary $\mathcal{A}$. The reduction forwards (m_0, m_1) to the IND-CPA challenger to obtain the challenge ciphertext ct_b, which can also be considered a challenge commitment in the hiding game. If the hiding adversary $\mathcal{A}$ can distinguish whether ct_b is a commitment to m_0 or m_1 with an advantage of $\epsilon_{\mathsf{hide}}(\lambda)$, then the reduction can also distinguish whether ct_b encrypts m_0 or m_1 with an advantage of at most $\epsilon_{\mathsf{IND\text{-}CPA}}(\lambda)$ by definition.

IV. Instantiations

This section describes several instantiations suitable for encryption schemes whose encryption algorithm can be represented as a Pedersen commitment computed from a multi-exponentiation operation. One existing candidate is an additively-homomorphic ElGamal encryption as ($\mathsf{ct} = \{\square^r \cdot \square^m, \square^r\}$). The decryption requires finding the discrete log of g^m, which restricts the message space to be short enough. Therefore, we split a large message $\mathcal{M}$ into short message spaces as $\mathcal{M} = (m_1 || \dots || m_l)$, and encrypt each block m_i in the form of $\mathsf{pk}^r \cdot g^{m_i}$. The decryptor, by removing the the blinding factor, can obtain m_i through simple brute-forcing. Since not all encryption schemes involve group multi-exponentiation, we cannot claim that ciphertexts from most encryption schemes can be compatible with Pedersen commitment; only *group-based encryption families* satisfy the condition. Therefore, we describe a method that becomes Pedersen $\mathsf{cc\text{-}Enc}$ by modifying well-known group-based encryption such as identity-, hierarchical identity-, and attribute-based encryption. Additionally, we formally provide the security proof for each scheme.

A. Identity based Encryption

Identity-Based Encryption (IBE) is a type of public key encryption that allows a user to encrypt and decrypt messages using a public key, which is derived from their identity information (id). The advantage lies in the simplification of the public key, eliminating the need for sharing a complex public key between users, as required in traditional public-key cryptography. There are several well-known Identity-Based Encryption (IBE) schemes, such as the Boneh-Boyen scheme [10] and the Waters scheme [18].

Our $\mathsf{cc\text{-}Enc}_{\mathsf{IBE}}$ based on [18]. In this part, we describe the process of transforming Waters' identity-based encryption scheme into $\mathsf{cc\text{-}Enc}_{\mathsf{IBE}}$. The scheme employs type-1 pairing and is designed based on the computational Diffie-Hellman assumption. Our ultimate objective is to modify the scheme into a format that can be used to Linker and made usable in the Tangram framework.

Guide for setup. In order to substitute type-1 pairing for type-3 pairing and shift the domain a ciphertext, we extend id related elements of pp (i.e. $\bar{u}$ and $\boldsymbol{U}$). By adjusting these elements to have the same exponent across different groups, additional elements are generated, leading to an increase in the size of the public parameter pp. Additionally, to transform the existing ciphertexts into a Pedersen-like form, an extra element is required. As a result, the size increases by 1 $\mathbb{G}_1$ and $l+1$ $\mathbb{G}_2$.

Guide for key generation. It is identical to the original algorithm except for shifting the elements to $\mathbb{G}_2$ and yielding a commitment key ck.

Guide for encryption. Recall that the form of the ciphertext ct_1 is represented as:

$$\mathsf{ct}_1 = e(g_1, h)^t \cdot m$$

We shift each ciphertext to the domain $\mathbb{G}_1$ and convert ct_1 into a Pedersen-like commitment. So, the original ciphertext ct_1 is modified as follows.

$$\mathsf{ct}_1 := g_2^t \cdot g_1^m$$

In addition, we can see ct_2 and ct_3 as the Pedersen commitment to the zero message.

Guide for decryption. It is similar to the original scheme, but the plaintext can be obtained by solving the discrete log (for small domain)[2].

In Fig.1, we describe our identity-based commit-carrying encryption scheme based on [18].

Security of our $\mathsf{cc\text{-}Enc}_{\mathsf{IBE}}$. The security of our $\mathsf{cc\text{-}Enc}_{\mathsf{IBE}}$ scheme, which stems from modifying the Waters' scheme, can be proven to be similar. We omit the detailed security proof for Waters' scheme and focus on providing the security proof specifically for our scheme. Before describing that our scheme is $\mathsf{cc\text{-}Enc}$, we show that our scheme is secure compared to Waters' scheme [18]. We use $weak$-EXDH (wEXDH)

[2]If we use the preprocessed table, the decryption time is more better.

```
$\mathbf{\Pi}_{\mathsf{ccIBE}}.\mathsf{Setup}(1^\lambda) \to (\mathsf{pp}, \mathsf{mk})$
$(g, h) \xleftarrow{\$} \mathbb{G}_1 \times \mathbb{G}_2$
$\alpha, \beta, \gamma \xleftarrow{\$} \mathbb{F}$
$g_1 \leftarrow g^\alpha, g_2 \leftarrow g^{\alpha\beta}$
$\boldsymbol{u} = \{u_i\}_{i=1}^n \xleftarrow{\$} \mathbb{F}^n$
$\bar{u} \leftarrow g^\gamma, \bar{v} \leftarrow h^\gamma$
$\boldsymbol{U} \leftarrow g^{\boldsymbol{u}} \in \mathbb{G}_1^n, \boldsymbol{V} \leftarrow h^{\boldsymbol{u}} \in \mathbb{G}_2^n$
$\mathsf{pp} := (g, h, g_1, g_2, \bar{u}, \boldsymbol{U}, \bar{v}, \boldsymbol{V})$
$\mathsf{mk} := h^{\alpha\beta}$
Return $(\mathsf{pp}, \mathsf{mk})$

$\mathbf{\Pi}_{\mathsf{ccIBE}}.\mathsf{KeyGen}(1^\lambda, \mathsf{pp}, \mathsf{id}, \mathsf{mk}) \to (\mathsf{ck}, \mathsf{pk}, \mathsf{sk})$
parse $\mathsf{pp}$ as $(g, h, g_1, g_2, \bar{u}, \boldsymbol{U}, \bar{v}, \boldsymbol{V})$
parse $\mathsf{id}$ as $(\mathsf{id}_1, \ldots, \mathsf{id}_n)$
$r \xleftarrow{\$} \mathbb{F}$
$X \leftarrow \bar{u} \cdot \prod_{i \in \mathcal{S}} U_i,\ W \leftarrow \bar{v} \cdot \prod_{i \in \mathcal{S}} V_i$
/ $\mathcal{S} := \forall i$: $\mathsf{id}_i$=1
$\mathsf{ck} := (g, g_1, g_2, X)$
$\mathsf{pk} := \mathsf{id}$
$\mathsf{sk} := (h^r, \mathsf{mk} \cdot W^r)$
Return $(\mathsf{ck}, \mathsf{pk}, \mathsf{sk})$

$\mathbf{\Pi}_{\mathsf{ccIBE}}.\mathsf{Enc}(\mathsf{pp}, \mathsf{pk}, m) \to (\mathsf{ct}; t)$
parse $\mathsf{pp}$ as $(g, h, g_1, g_2, \bar{u}, \boldsymbol{U}, \bar{v}, \boldsymbol{V})$
parse $\mathsf{pk}$ as $(\mathsf{id}_1, \ldots, \mathsf{id}_n)$
$t \xleftarrow{\$} \mathbb{F}$
$\mathsf{ct}_1 \leftarrow g_2^t \cdot g_1^m$ , $\mathsf{ct}_2 \leftarrow g^t$
$\mathsf{ct}_3 \leftarrow \left(\bar{u} \cdot \sum_{i \in \mathcal{S}} U_i\right)^t$   / $\mathcal{S} := \forall i$: $\mathsf{id}_i$=1
$\mathsf{ct} := (\mathsf{ct}_1, \mathsf{ct}_2, \mathsf{ct}_3)$
Return $(\mathsf{ct}; t)$

$\mathbf{\Pi}_{\mathsf{ccIBE}}.\mathsf{Dec}(\mathsf{pp}, \mathsf{sk}, \mathsf{ct}) \to m$
parse $\mathsf{pp}$ as $(g, h, g_1, g_2, \bar{u}, \boldsymbol{U}, \bar{v}, \boldsymbol{V})$
parse $\mathsf{sk}$ as $(\mathsf{sk}_1, \mathsf{sk}_2)$
parse $\mathsf{ct}$ as $(\mathsf{ct}_1, \mathsf{ct}_2, \mathsf{ct}_3)$
$e(\mathsf{ct}_1, h) \cdot \frac{e(\mathsf{ct}_3, \mathsf{sk}_1)}{e(\mathsf{ct}_2, \mathsf{sk}_2)} = e(g_1, h)^m$
/ Compute a discrete log to obtain $m$
Return $m$

$\mathbf{\Pi}_{\mathsf{ccIBE}}.\mathsf{VerEnc}(\mathsf{ck}, \mathsf{ct}, t, m) \to \mathsf{true/false}$
parse $\mathsf{ck}$ as $(g, g_1, g_2, X)$
parse $\mathsf{ct}$ as $(\mathsf{ct}_1, \mathsf{ct}_2, \mathsf{ct}_3)$
return $\mathsf{ct}_1 \overset{?}{=} g_2^t \cdot g_1^m \wedge \mathsf{ct}_2 \overset{?}{=} g^t \wedge \mathsf{ct}_3 \overset{?}{=} X^t$
```

Figure 1: Our $\mathsf{cc\text{-}Enc}_{\mathsf{IBE}}$ based on [18]

assumption for an asymmetric paring $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_t$ is stated in [48, 49].

Definition 4 (wEXDH Assumption)**.** Given $(g, g^\alpha, g^{\alpha\beta}, g^\gamma) \in \mathbb{G}_1^4$ and $(h, h^\alpha, h^\beta) \in \mathbb{G}_2^3$, along with $g^z \in \mathbb{G}_1$, it is hard to distinguish whether $z = \alpha \cdot \beta \cdot \gamma$ or z is random except for advantage ϵ.

Theorem 1. The construction in figure 1 is IND-CPA secure if wEXDH assumption holds.

Proof. Assume that there exists an adversary, denoted as $\mathcal{A}$, with an advantage ϵ attacking the IND-CPA security within our proposed scheme. Our aim is to construct that an adversary, with equivalent advantage ϵ, could feasibly exist within the framework of the wEXDH assumption. We first define a simulator Sim to play the wEXDH game. This simulator is presented with an wEXDH challenge defined by the tuple $(g, g_1 = g^\alpha, g_2 = g^{\alpha\beta}, g_3 = g^\gamma, h, h_1 = h^\alpha, h_2 = h^\beta, g^z)$. Sim outputs 1 if $z = \alpha\beta\gamma$, otherwise 0. The simulator runs $\mathcal{A}$ that works as below.

Setup phase. Sim chooses a random integer m and k from the set $\{0, \cdots, n\}$ such that $p > mn$. The simulator then chooses random elements x' from $\{0, \cdots, m-1\}$, y' from the group $\mathbb{Z}_p$. Additionally, Sim generates two random vectors, $\{x_i\}_{i=1}^n$ and $\{y_i\}_{i=1}^n$, each of length n, with their elements randomly chosen from $\{0, \cdots, m-1\}$ and the group $\mathbb{Z}_p$. After choosing these values, Sim constructs the public parameters, which are sent to the adversary $\mathcal{A}$. The public parameters, denoted as pp, are constructed as follows:

$$\mathsf{pp} = \begin{pmatrix} g, h, g_1 = g^\alpha, g_2 = g^{\alpha\beta}, \\ \bar{u} = g_1^{p-km+x'} g^{y'}, \bar{v} = h_1^{p-km+x'} h^{y'}, \\ \boldsymbol{U} = \{g_1^{x_i} g^{y_i}\}_{i=1}^n, \boldsymbol{V} = \{h_1^{x_i} h^{y_i}\}_{i=1}^n \end{pmatrix}.$$

We will define some additional functions to simplify the representation of $\bar{u}$, $\bar{v}$, $\boldsymbol{U}$ and $\boldsymbol{V}$. Let $\mathcal{V} \subseteq \{1, \cdots, n\}$ be the set of all i for which $\nu_i = 1$ for an identity ν. We define two functions $X(\nu) = p - km + x' + \sum_{i \in \mathcal{V}} x_i$ and $Y(\nu) = y' + \sum_{i \in \mathcal{V}} y_i$ that represent the linear combination of each exponent. Additionally, we will define a binary function $K(\nu)$ as

$$K(\nu) = \begin{cases} 0, & \text{if } x' + \sum_{i \in \mathcal{V}} x_i = 0 \bmod m \\ 1, & \text{otherwise} \end{cases}$$

Phase 1. $\mathcal{A}$ will issue secret key query for an identity ν. Sim chooses a random $r \in \mathbb{Z}_p$ and constructs the secret key d using the technique in [10]

$$d = (d_0, d_1) = \left(h_2^{-\frac{Y(\nu)}{X(\nu)}} \left(\bar{v} \prod_{i \in \mathcal{V}} V_i \right)^r, h_2^{-\frac{1}{X(\nu)}} h^r \right).$$

Let $\tilde{r} = r - \frac{\beta}{X(\nu)}$. Then we have

$$d_0 = h_2^{-\frac{Y(\nu)}{X(\nu)}} \left(\bar{v} \prod_{i \in \mathcal{V}} V_i \right)^r = h_2^{-\frac{Y(\nu)}{X(\nu)}} (h_1^{X(\nu)} h^{Y(\nu)})^r$$

$$= h_1^\beta (h_1^{X(\nu)} h^{Y(\nu)})^{-\frac{\beta}{X(\nu)}} (h_1^{X(\nu)} h^{Y(\nu)})^r = h_1^\beta \left(\bar{v} \prod_{i \in \mathcal{V}} V_i \right)^{\tilde{r}},$$

$$d_1 = h_2^{-\frac{1}{X(\nu)}} h^r = h^{r - \frac{\beta}{X(\nu)}} = h^{\tilde{r}}.$$

Sim can computes the above if and only if $X(\nu) \neq 0 \bmod p$. If $K(\nu) = 1$, $x' + \sum_{i \in \mathcal{V}} x_i \neq 0 \bmod m$, which is $x' + \sum_{i \in \mathcal{V}} x_i - km \neq 0$ for all $k \in \{0, \cdots, n\}$. Thus, we have $X(\nu) = p - km + x' + \sum_{i \in \mathcal{V}} x_i = p + (x' + \sum_{i \in \mathcal{V}} x_i - km) \neq 0 \bmod p$, the simulator will not abort in a sufficient condition where $K(\nu) = 1$.

Challenge phase. $\mathcal{A}$ will send two messages $M_0, M_1 \in \mathbb{Z}_p$ and an identity ν^*. If $x' + \sum_{i \in \mathcal{V}^*} x_i \neq km$, $X(\nu^*) \neq 0 \bmod p$ and Sim cannot construct ct. Thus, Sim will abort and guess the random for wEXDH challenge. Otherwise, Sim will flip a public coin b and construct the ciphertext as $\mathsf{ct} = (Zg_1^{M_b}, C, C^{Y(\nu^*)})$ where $Z = g^z$ and $C = g_3$. If $Z = g^{\alpha\beta\gamma}$, then we have

$$\mathsf{ct} = \left(g^{\alpha\beta\gamma}g_1^{M_b}, g^\gamma, g^{\gamma Y(\nu^*)}\right) = \left(g_2^\gamma g_1^{M_b}, g^\gamma, \left(\bar{u}\prod_{i\in\mathcal{V}^*} U_i\right)^\gamma\right).$$

Under these conditions, ct is a valid ciphertext of M_b. If it does not hold, it implies that Z is a random element of $\mathbb{G}_1$. This means that ct does not leak any information about the simulator's choice of b.

Phase 2. $\mathcal{A}$ repeats the same method to issue private key.

Guess. Finally, $\mathcal{A}$ outputs a guess b' of b. If $b = b'$ then Sim outputs 1, that is equivalent to Z being $g^{\alpha\beta\gamma}$. Otherwise, Sim outputs 0, implying that Z is a random element within $\mathbb{Z}_p$.

The event that Sim does not abort is $(\bigwedge_{i=1}^q K(\nu_i) = 1) \wedge x' + \sum_{i\in\mathcal{V}^*} x_i = km$ for q key queries in Phase 1 and Phase 2. In other words, there does not exist ν_i such that $K(\nu_i) = 0$ for q key queries, and it must satisfy $K(\nu^*) = 0$, in other words, there exists $k \in \{0, \cdots, n\}$ such that $x' + \sum_{i\in\mathcal{V}^*} x_i = km$. The probability that the above event occurs is

$$\begin{aligned}
&\Pr\left[\left(\bigwedge_{i=1}^q K(\nu_i) = 1\right) \wedge x' + \sum_{i\in\mathcal{V}^*} x_i = km\right] \\
&= \left(1 - \Pr\left[\bigvee_{i=1}^q K(\nu_i) = 0\right]\right) \Pr\left[x' + \sum_{i\in\mathcal{V}^*} x_i = km | \bigwedge_{i=1}^q K(\nu_i) = 1\right] \\
&\geq \left(1 - \sum_{i=1}^q \Pr[K(\nu_i) = 0]\right) \Pr\left[x' + \sum_{i\in\mathcal{V}^*} x_i = km | \bigwedge_{i=1}^q K(\nu_i) = 1\right] \\
&= \left(1 - \frac{q}{m}\right) \Pr\left[x' + \sum_{i\in\mathcal{V}^*} x_i = km | \bigwedge_{i=1}^q K(\nu_i) = 1\right] \\
&= \left(1 - \frac{q}{m}\right) \frac{1}{n+1} \Pr\left[K(\nu^*) = 0 \middle| \bigwedge_{i=1}^q K(\nu_i) = 1\right] \\
&= \left(1 - \frac{q}{m}\right) \frac{1}{n+1} \frac{\Pr[K(\nu^*) = 0]}{\Pr[\bigwedge_{i=1}^q K(\nu_i) = 1]} \Pr\left[\bigwedge_{i=1}^q K(\nu_i) = 1 | K(\nu^*) = 0\right] \\
&\geq \left(1 - \frac{q}{m}\right) \frac{1}{(n+1)m} \Pr\left[\bigwedge_{i=1}^q K(\nu_i) = 1 \middle| K(\nu^*) = 0\right] \\
&= \left(1 - \frac{q}{m}\right) \frac{1}{(n+1)m} \left(1 - \Pr\left[\bigvee_{i=1}^q K(\nu_i) = 0 | K(\nu^*) = 0\right]\right) \\
&\geq \left(1 - \frac{q}{m}\right) \frac{1}{(n+1)m} \left(1 - \sum_{i=1}^q \Pr[K(\nu_i) = 0 | K(\nu^*) = 0]\right) \\
&= \left(1 - \frac{q}{m}\right)^2 \frac{1}{(n+1)m} \geq \left(1 - \frac{2q}{m}\right) \frac{1}{(n+1)m}.
\end{aligned}$$

After the abortion, we can now see $\mathcal{A}$'s view where the input tuple is sampled from. If the input tuple is sampled from where $z = \alpha\beta\gamma$, the $\mathcal{A}$'s view is equivalent to a real attack game. In this case, $\mathcal{A}$ must satisfy $|\Pr[b = b'] - \frac{1}{2}| \geq \epsilon$. On the other hands, the input tuple sampled from where z is uniformly random means that $\Pr[b = b'] = \frac{1}{2}$. Therefore, with uniformly chosen $\alpha, \beta, \gamma \in \mathbb{Z}_p$, $g \in \mathbb{G}_1$, and $h \in \mathbb{G}_2$, we have that

$$\begin{aligned}
&|\Pr\left[\mathsf{Sim}(g, h, g^\alpha, g^{\alpha\beta}, g^\gamma, h^\alpha, h^\beta, g^{\alpha\beta\gamma}) = 0\right] \\
&- \Pr\left[\mathsf{Sim}(g, h, g^\alpha, g^{\alpha\beta}, g^\gamma, h^\alpha, h^\beta, g^z) = 0\right]| \\
&\geq \left|\frac{1}{2} + \epsilon - \frac{1}{2}\right| = \epsilon
\end{aligned}$$

It means that Sim can distinguish g^z with the advantage ϵ where ϵ is the same as $\mathcal{A}$'s advantage.

$\square$

B. Hierarchical identity based Encryption

Hierarchical Identity-Based Encryption (HIBE) is a type of IBE that allows users to encrypt and decrypt messages using their hierarchical identity information. In HIBE, each user is associated with a hierarchical identity, which consists of a set of attributes that represent different levels of authority. The advantage of HIBE is that it provides a flexible way of controlling access to encrypted data based on the hierarchical structure of an organization. There are several well-known HIBE schemes, such as [11, 50]. The security of HIBE schemes is typically based on the computational DH assumption or the bilinear DH assumption.

Our $\mathsf{cc\text{-}Enc_{HIBE}}$ **based on [11]**. In this part, we describe the process of transforming Boneh-Boyen HIBE scheme into $\mathsf{cc\text{-}Enc_{HIBE}}$. Similar to the previous IBE scheme, this scheme employs type-1 pairing and is designed based on the decisional Diffie-Hellman assumption. Therefore, our ultimate objective is the same as in $\mathsf{cc\text{-}Enc_{IBE}}$. As mentioned before, the part ct_1 can be transformed into an additively-homomorphic ElGamal-like ciphertext, while the parts $(\mathsf{ct}_2, \mathsf{ct}_3)$ can be linked to Linker. Likewise, we extend this scheme to $\mathsf{cc\text{-}Enc_{HIBE}}$ using type-3 pairing.

Guide for setup. As with $\mathsf{cc\text{-}Enc_{IBE}}$, we extend id related elements of pp to use type-3 pairing. The elements in $\mathbb{G}_2$ are generated by having the same exponent about id (i.e. $\bar{v}$ and $\boldsymbol{V}$). We additionally require an extra element to modify the ciphertexts to resemble a Pedersen commitment. As the result of the added elements, the size of pp increases by 1 $\mathbb{G}_1$ and $\ell + 1$ $\mathbb{G}_2$.

Guide for key generation. It follows the original algorithm except for shifting the elements to $\mathbb{G}_2$ and yielding a commitment key ck.

Guide for encryption. The ciphertext ct_1 in [11] is composed of

$$\mathsf{ct}_1 = e(g^\alpha, h)^t \cdot m.$$

We shift the ciphertext to the domain $\mathbb{G}_1^3$ and modify the ciphertext into a Pedersen commitment. The ciphertext ct_1 is modified as follows.

$$\mathsf{ct}_1 := g_2^\alpha \cdot g_1^m$$

Guide for decryption. It is similar to the original scheme, but because we modify the ciphertext into a Pedersen-like commitment, we have to compute one more pairing for decryption. Additionally it is required to solve the small domain discrete log problem to obtain the plaintext.

In Fig.2, we describe our hierarchical identity-based commit-carrying encryption scheme based on [11].

Security of our $\mathsf{cc\text{-}Enc_{HIBE}}$. To prove the property of $\mathsf{cc\text{-}Enc_{HIBE}}$, we use a slightly stronger assumption ℓ-wEXDH [49] extended from wEXDH [48].

```
$\mathbf{\Pi}_{\mathsf{ccHIBE}}.\mathsf{Setup}(1^\lambda, \ell) \to (\mathsf{pp}, \mathsf{mk})$
    $(g, h) \xleftarrow{\$} \mathbb{G}_1 \times \mathbb{G}_2$
    $\alpha, \beta, \gamma \xleftarrow{\$} \mathbb{F}$
    $\boldsymbol{u} = \{u_i\}_{i=1}^{\ell} \xleftarrow{\$} \mathbb{F}^\ell, \bar{u} \leftarrow g^\gamma, \bar{v} \leftarrow h^\gamma$
    $\boldsymbol{U} \leftarrow g^{\boldsymbol{u}} \in \mathbb{G}_1^\ell, \boldsymbol{V} \leftarrow h^{\boldsymbol{u}} \in \mathbb{G}_2^\ell$
    $\mathsf{pp} := (g, h, g^\alpha, g^{\alpha\beta}, \boldsymbol{U}, \boldsymbol{V}, \bar{u}, \bar{v})$
    $\mathsf{mk} := h^{\alpha\beta}$
    Return $(\mathsf{pp}, \mathsf{mk})$

$\mathbf{\Pi}_{\mathsf{ccHIBE}}.\mathsf{KeyGen}(\mathsf{pp}, \mathsf{mk}, \mathsf{ID}_k) \to (\mathsf{ck}, \mathsf{pk}, \mathsf{sk})$
    parse $\mathsf{pp}$ as $(g, h, g_1, g_2, \boldsymbol{U}, \boldsymbol{V}, \bar{u}, \bar{v})$
    parse $\mathsf{ID}_k$ as $(\mathsf{id}_1, \dots, \mathsf{id}_k)$
    $r \xleftarrow{\$} \mathbb{F}$
    $X \leftarrow \bar{u} \cdot \prod_{i \in k} U_i^{\mathsf{id}_i}, W \leftarrow \bar{v} \cdot \prod_{i \in k} V_i^{\mathsf{id}_i}$
    $\mathsf{ck} := (g, g_1, g_2, X)$
    $\mathsf{pk} := \mathsf{ID}_k$
    $\mathsf{sk} := (h^r, \mathsf{mk} \cdot W^r, V_{k+1}^r, \dots, V_\ell^r)$
    Return $(\mathsf{ck}, \mathsf{pk}, \mathsf{sk})$

$\mathbf{\Pi}_{\mathsf{ccHIBE}}.\mathsf{Enc}(\mathsf{pp}, \mathsf{pk}, m) \to (\mathsf{ct}; t)$
    parse $\mathsf{pp}$ as $(g, h, g_1, g_2, \boldsymbol{U}, \boldsymbol{V}, \bar{u}, \bar{v})$
    parse $\mathsf{pk}$ as $(\mathsf{id}_1, \dots, \mathsf{id}_k)$
    $t \xleftarrow{\$} \mathbb{F}$
    $\mathsf{ct}_1 \leftarrow g_2^t \cdot g_1^m$ , $\mathsf{ct}_2 \leftarrow g^t$
    $\mathsf{ct}_3 \leftarrow \left(\bar{u} \cdot \prod_{i \in k} U_i^{\mathsf{id}_i}\right)^t$
    $\mathsf{ct} := (\mathsf{ct}_1, \mathsf{ct}_2, \mathsf{ct}_3)$
    Return $(\mathsf{ct}; t)$

$\mathbf{\Pi}_{\mathsf{ccHIBE}}.\mathsf{Dec}(\mathsf{pp}, \mathsf{sk}, \mathsf{ct}) \to m$
    parse $\mathsf{pp}$ as $(g, h, g_1, g_2, \boldsymbol{U}, \boldsymbol{V}, \bar{u}, \bar{v})$
    parse $\mathsf{sk}$ as $(\mathsf{sk}_1, \mathsf{sk}_2)$
    parse $\mathsf{ct}$ as $(\mathsf{ct}_1, \mathsf{ct}_2, \mathsf{ct}_3)$
    $e(\mathsf{ct}_1, h) \cdot \frac{e(\mathsf{ct}_3, \mathsf{sk}_1)}{e(\mathsf{ct}_2, \mathsf{sk}_2)} = e(g_1, h)^m$ / Compute a discrete log to obtain m
    Return $m$

$\mathbf{\Pi}_{\mathsf{ccHIBE}}.\mathsf{VerEnc}(\mathsf{ck}, \mathsf{ct}, t, m) \to \mathsf{true}/\mathsf{false}$
    parse $\mathsf{ck}$ as $(g, g_1, g_2, X)$
    parse $\mathsf{ct}$ as $(\mathsf{ct}_1, \mathsf{ct}_2, \mathsf{ct}_3)$
    return $\mathsf{ct}_1 \overset{?}{=} g_2^t \cdot g_1^m \wedge \mathsf{ct}_2 \overset{?}{=} g^t \wedge \mathsf{ct}_3 \overset{?}{=} X^t$
```

Figure 2: Our cc-Enc$_{\mathsf{HIBE}}$ based on [11]

Definition 5 (ℓ-wEXDH Assumption)**.** Given $(g, g^\alpha, \dots, g^{\alpha^{\ell+1}}, g^\beta, g^z) \in \mathbb{G}_1^{\ell+4}$ and $(h, h^\alpha, \dots, h^{\alpha^\ell}) \in \mathbb{G}_2^{\ell+1}$, it is hard to distinguish whether $z = \alpha^{\ell+1} \cdot \beta$ or z is random.

The reduction between wEXDH assumption and ℓ-wEXDH assumption is tight as in [49]. For asserting security within a hierarchy of depth ℓ, we lean on the ℓ-wEXDH assumption. Note that the ℓ-wEXDH assumption was formerly presented with an alternate label in [49], referred to as the $\mathcal{P}^\ell$-BDH assumption.

Theorem 2. Let $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ be a bilinear group of prime order p. Suppose the (Decision) (t, ϵ, ℓ)-wEXDH assumption holds in $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$, which means that there is no t-time algorithm with the advantage ϵ in solving the (Decision) ℓ-wEXDH problem. Then, our ℓ-cc-Enc$_{\mathsf{HIBE}}$ is $(t, q_{\mathsf{ID}}, \ell)$-selective identity, chosen plaintext (IND-sID-CPA) secure for arbitrary ℓ and q_{ID} private key queries.

Proof. We describe a formal security proof of our cc-Enc$_{\mathsf{HIBE}}$, which is similar to the method of [11]. Assume that an adversary $\mathcal{A}$ has advantage ϵ in attacking the ℓ-cc-Enc$_{\mathsf{HIBE}}$ system in which $\mathcal{A}$ uses an algorithm $\mathcal{B}$ that solves the ℓ-wEXDH problem.

Consider $y_i = g^{\alpha^i} \in \mathbb{G}_1$ and $\hat{y}_i = h^{\alpha^i} \in \mathbb{G}_2$ where g is a generator of $\mathbb{G}_1$, h is a generator of $\mathbb{G}_2$, and both α and β are in $\mathbb{Z}_p^*$. The algorithm $\mathcal{B}$ receives as input random tuples $(g, y_1, \dots, y_{\ell+1}, g^\beta, g^z)$ and $(h, \hat{y}_1, \dots, \hat{y}_\ell)$. These tuples can be sampled from where $z = \alpha^{\ell+1} \cdot \beta$ or from where z is random. The objective of $\mathcal{B}$ is to return 1 if the input tuples correspond to the condition $z = \alpha^{\ell+1} \cdot \beta$, and to return 0 if not. The strategy of $\mathcal{B}$ involves interaction with $\mathcal{A}$ through a selective identity game as detailed:

Initialization. In the selective identity game, $\mathcal{A}$ starts by specifying a target identity $\mathsf{ID}^* = (\mathrm{I}_1^*, \dots, \mathrm{I}_m^*) \in (\mathbb{Z}_p^*)^m$, intending to attack this identity with a constraint that $m \leq \ell$. If $m < \ell$, $\mathcal{B}$ appends $\ell - m$ zeros at the tail of ID^*, effectively transforming it into a ℓ-length vector. Henceforth, we regard ID^* as a ℓ-element vector.

Setup. For setting up the public parameter, $\mathcal{B}$ randomly selects a random $\beta \in \mathbb{Z}_p$ and sets $g_1 = y_1$ and $g_2 = y_{\ell+1}^\beta = g^{\alpha^{\ell+1} \cdot \beta}$, simultaneously setting $h_1 = \hat{y}_\ell^\beta = h^{\alpha^\ell \cdot \beta}$. Then, $\mathcal{B}$ randomly picks $\gamma_1, \dots, \gamma_\ell \in \mathbb{Z}_p$ and sets $\boldsymbol{U}_i = g^{\gamma_i}/y_{\ell-i+1}$, similarly setting $\boldsymbol{V}_i = h^{\gamma_i}/\hat{y}_{\ell-i+1}$ for $i = 1, \dots, \ell$. $\mathcal{B}$ also chooses an arbitrary random $\delta \in \mathbb{Z}_p$ and sets $\bar{u} = g^\delta \cdot \prod_{i=1}^{\ell} y_{\ell-i+1}^{\mathrm{I}_i^*}$ and $\bar{v} = h^\delta \cdot \prod_{i=1}^{\ell} \hat{y}_{\ell-i+1}^{\mathrm{I}_i^*}$. Ultimately, the public parameter $\mathsf{pp} = (g, h, g_1, g_2, \bar{u}, \bar{v}, \boldsymbol{U}_1, \dots, \boldsymbol{U}_\ell, \boldsymbol{V}_1, \dots, \boldsymbol{V}_\ell)$ is sent to $\mathcal{A}$. The master key tied to this public parameter is $h_1^\alpha = h^{\alpha^{\ell+1} \cdot \beta}$, but this cannot be computed by $\mathcal{B}$.

Phase 1. Consider a query for the private key query associated with $\mathsf{ID} = (\mathrm{I}_1, \dots, \mathrm{I}_u) \in (\mathbb{Z}_p^*)^u$ where $u \leq \ell$. It is imperative that ID is neither identical to ID^* nor a prefix thereof. This precondition guarantees a distinct $k \in \{1, \dots, u\}$ where $\mathrm{I}_k \neq \mathrm{I}_k^*$. If this is not the case, ID would serve as a prefix of ID^*. The index k is recognized as the lowest index meeting the precondition. To address the query, $\mathcal{B}$ first deduces a private key corresponding to the identity $(\mathrm{I}_1, \dots, \mathrm{I}_k)$. Subsequently, it constructs the private key related to the inquired identity $\mathsf{ID} = (\mathrm{I}_1, \dots, \mathrm{I}_k, \dots, \mathrm{I}_u)$.

For the creation of the private key associated with identity $(\mathrm{I}_1, \dots, \mathrm{I}_k)$, $\mathcal{B}$ commences by randomly choosing a random $\tilde{r} \in \mathbb{Z}_p$. The randomness r is defined as

$$r = \left(\frac{\alpha^k}{(\mathrm{I}_k - \mathrm{I}_k^*)} + \tilde{r}\right) \cdot \beta \in \mathbb{Z}_p.$$

Following this, $\mathcal{B}$ proceeds to construct the private key,

$$\left(h^r, h_1^\alpha \cdot (\bar{v}\boldsymbol{V}_1^{\mathrm{I}_1}\cdots\boldsymbol{V}_k^{\mathrm{I}_k})^r, \boldsymbol{V}_{k+1}{}^r, \ldots, \boldsymbol{V}_\ell{}^r\right), \quad (1)$$

that is tailored to match the private key corresponding to the identity $(\mathrm{I}_1, \ldots, \mathrm{I}_k)$. We demonstrate that using the values at its disposal, $\mathcal{B}$ can successfully compute each element of this private key. To deduce the initial part of the private key, it's observed that $(\bar{v}\boldsymbol{V}_1^{\mathrm{I}_1}\cdots\boldsymbol{V}_k^{\mathrm{I}_k})^r$ is equal to

$$\left(h^{\delta+\sum_{i=1}^k \mathrm{I}_i\gamma_i}\cdot\prod_{i=1}^{k-1}\hat{y}_{\ell-i+1}^{\mathrm{I}_i^*-\mathrm{I}_i}\cdot\hat{y}_{\ell-k+1}^{\mathrm{I}_k^*-\mathrm{I}_k}\cdot\prod_{i=k+1}^{\ell}\hat{y}_{\ell-i+1}^{\mathrm{I}_i^*}\right)^r. \quad (2)$$

As in [11], let Z denote the product of the first, second, and fourth terms. More precisely,

$$Z=\left(h^{\delta+\sum_{i=1}^k \mathrm{I}_i\gamma_i}\cdot\underbrace{\prod_{i=1}^{k-1}\hat{y}_{\ell-i+1}^{\mathrm{I}_i^*-\mathrm{I}_i}}_{=1}\cdot\prod_{i=k+1}^{\ell}\hat{y}_{\ell-i+1}^{\mathrm{I}_i^*}\right)^r.$$

Because the second term in Z equals 1, Z can be calculated using the values that $\mathcal{B}$ has. So what is left is third term in Eq (3), namely $\hat{y}_{\ell-k+1}^{r(\mathrm{I}_k^*-\mathrm{I}_k)}$, simplifies to

$$\hat{y}_{\ell-k+1}^{r(\mathrm{I}_k^*-\mathrm{I}_k)}=\left(\hat{y}_{\ell-k+1}^{\tilde{r}(\mathrm{I}_k^*-\mathrm{I}_k)}\cdot\hat{y}_{\ell-k+1}^{(\mathrm{I}_k^*-\mathrm{I}_k)\frac{\alpha^k}{\mathrm{I}_k-\mathrm{I}_k^*}}\right)^\beta=\left(\hat{y}_{\ell-k+1}^{\tilde{r}(\mathrm{I}_k^*-\mathrm{I}_k)}/\hat{y}_{\ell+1}\right)^\beta.$$

Consequently, the second component in the private key (2) becomes:

$$h_1^\alpha\cdot(\bar{v}\boldsymbol{V}_1^{\mathrm{I}_1}\cdots\boldsymbol{V}_k^{\mathrm{I}_k})^r=\hat{y}_{\ell+1}^\beta\cdot Z\cdot\left(\hat{y}_{\ell-k+1}^{\tilde{r}(\mathrm{I}_k^*-\mathrm{I}_k)}/\hat{y}_{\ell+1}\right)^\beta=Z\cdot\hat{y}_{\ell-k+1}^{\tilde{r}(\mathrm{I}_k^*-\mathrm{I}_k)\beta}$$

Given that $\hat{y}_{\ell+1}$ is nullified in the equation, every term in the mentioned expression is known to $\mathcal{B}$. Therefore, $\mathcal{B}$ can calculate the first component of the private key.

The first component, which is denoted by h^r, it can be calculated as $\left(\hat{y}_k^{1/(\mathrm{I}_k-\mathrm{I}_k^*)}\cdot h^{\tilde{r}}\right)^\beta$. In a similar way, the subsequent components, namely $\boldsymbol{V}_{k+1}{}^r, \ldots, \boldsymbol{V}_\ell{}^r$, can be calculated by $\mathcal{B}$. As a result, $\mathcal{B}$ can compute a legitimate private key corresponding to identity $(\mathrm{I}_1, \ldots, \mathrm{I}_k)$. $\mathcal{B}$ leverages the derived private key to compute corresponding for the descendant identity ID and gives $\mathcal{A}$ the key.

Challenge phase. After the completion of Phase 1, $\mathcal{A}$ outputs two messages M_0, M_1. In response, $\mathcal{B}$ randomly determines a bit $b\in\{0,1\}$ and outputs the challenge ciphertext, denoted as:

$$\mathcal{CT}=\left(g^{z\gamma}\cdot g_1{}^{M_b}, g^\gamma, (g^\gamma)^{\delta+\sum_{i=1}^k\mathrm{I}_i^*\gamma_i}\right).$$

The terms g^γ and g^z are directly from the tuple that was previously presented to $\mathcal{B}$. And

$$\begin{aligned}(g^\gamma)^{\delta+\sum_{i=1}^k\mathrm{I}_i^*\gamma_i}&=\left(\prod_{i=1}^{\ell}(g^{\gamma_i}/y_{\ell-i+1})^{\mathrm{I}_i^*}\cdot(g^\delta\prod_{i=1}^{\ell}y_{\ell-i+1})^{\mathrm{I}_i^*})\right)^\gamma\\&=(\bar{u}\boldsymbol{U}_1^{\mathrm{I}_1^*}\cdots\boldsymbol{U}_\ell^{\mathrm{I}_\ell^*})^\gamma,\\g^{\alpha^{\ell+1}\beta\gamma}&=g_2{}^\gamma.\end{aligned}$$

Given the premise that $z=\alpha^{\ell+1\cdot\beta}$, the challenge ciphertext is a legitimate encryption of M_b with respect to the identity $\mathsf{ID}^*=(\mathrm{I}_1^*, \ldots, \mathrm{I}_m^*)$ that was selected by $\mathcal{A}$, since

$$\begin{aligned}CT&=\left(g_2{}^\gamma\cdot g_1{}^{M_b}, g^\gamma, (\boldsymbol{U}_1^{\mathrm{I}_1^*}\cdots\boldsymbol{U}_m^{\mathrm{I}_m^*}\cdots\boldsymbol{U}_\ell^{\mathrm{I}_\ell^*})^\gamma\right)\\&=\left(g_2{}^\gamma\cdot g_1{}^{M_b}, g^\gamma, (\boldsymbol{U}_1^{\mathrm{I}_1^*}\cdots\boldsymbol{U}_m^{\mathrm{I}_m^*})^\gamma\right)\end{aligned}$$

Otherwise, when z is uniformly random, $\mathcal{CT}$ is independent of b in the $\mathcal{A}$'s view.

Phase 2. $\mathcal{A}$ issues up to q_{ID} queries not issued in Phase 1. $\mathcal{B}$ responds in the same manner as previously.

Guess. $\mathcal{A}$ outputs a guess $b'\in\{0,1\}$. With this guess in hand, $\mathcal{B}$ concludes its own game based on the following criteria: If $b=b'$ then $\mathcal{B}$ outputs 1. This output indicates that z corresponds to $\alpha^{\ell+1}\cdot\beta$. On the contrary, if $b\neq b'$ then $\mathcal{B}$ outputs 0. This implies that z is a random element within $\mathbb{Z}_p$.

When the input tuple is sampled from where $z=\alpha^{\ell+1}\cdot\beta$, the perspective of $\mathcal{A}$ is aligns with its perspective in a real attack game. As a result, $\mathcal{A}$ meets $|\Pr[b=b']-\frac{1}{2}|\geq\epsilon$. Conversely, if the input tuple is sampled from where z is random, then $\Pr[b=b']=\frac{1}{2}$. Given that g and h are uniformly distributed in $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively, and α, β and z are uniformly distributed in $\mathbb{Z}_p$, it follows that the difference in probabilities:

$$\begin{aligned}|\Pr[\mathcal{B}(g,h,\boldsymbol{y}_{\ell+1},\hat{\boldsymbol{y}}_\ell,g^\beta,g^{\alpha^{\ell+1}\beta})=0]\\-\Pr[\mathcal{B}(g,h,\boldsymbol{y}_{\ell+1},\hat{\boldsymbol{y}}_\ell,g^\beta,g^z)=0]|\geq\epsilon.\end{aligned}$$

This outcome aligns with the desired result, thereby finalizing the proof of the theorem. □

C. Attribute Based Encryption

Intuitively, Attribute-Based Encryption (ABE) is a type of encryption that enables access control to encrypted data based on attributes or policies associated with the data. In ABE, data is encrypted using an access policy that specifies a set of attributes. The attributes can be anything from user roles to personal identifiers such as age or location, or even arbitrary strings. It ensures that only users who possess the necessary attributes can decrypt the data, based on the access policy specified by the data owner. There are many types of ABE schemes such as ciphertext-policy ABE and key-policy ABE. One distinguishing factor between these schemes is the location where the access policy is embedded. We focus on the ciphertext-policy variant of ABE and extend it to $\mathsf{cc\text{-}Enc_{ABE}}$.

Our $\mathsf{cc\text{-}Enc_{ABE}}$ based on [12]. We redefine the hash function $H:\{0,1\}^*\to\mathbb{F}$, since our scheme uses type-3 pairing. We denote a hash table as HT, which is a collection of group elements computed using the generators g and h, where each element is the result of applying the hash function H to $\mathsf{attr}(x)_{x\in\mathsf{ATree}}$. Hence, each element is constructed as a tuple $(H_1(\mathsf{attr}(x)), H_2(\mathsf{attr}(x))) := (g^{H(\mathsf{attr}(x))}, h^{H(\mathsf{attr}(x))})$.

Guide for setup. In a circuit, it is hard to check all the number of nodes in the access tree. For this reason, we extend ATree of pp to the hash table HT. In addition, we need to shift the domain of some elements to $\mathbb{G}_2$ for using type-3 pairing. As a result, the size of pp increases by the size of the access tree and by one $\mathbb{G}_1$ element for the Pedersen commitment.

Guide for encryption. Since we use the hash table HT rather than checking all the nodes in the access tree, we can easily find $H_1(\mathsf{attr}(x))$ in HT. We also modify ct_2 to the Pedersen commitment as follows:

$$\mathsf{ct}_2 := g_2^s g_3^m$$

Guide for key generation. It follows the original scheme except for shifting the elements to $\mathbb{G}_2$ and yielding a commitment key ck.

Guide for decryption. It is similar to the original scheme, but the difference is that we obtain the plaintext by solving the small domain discrete logarithm.

In Fig.3, we illustrate our commit-carrying attribute-based encryption scheme based on [12].

Security of our $\mathsf{cc\text{-}Enc_{ABE}}$. We show that no efficient adversary working generically on the groups underlying our scheme can break the security of our scheme with non-negligible probability. We use the generic bilinear group model and the random oracle model to prove this case. The generic group model (GGM) was introduced to reason about lower bounds for computing the family of discrete logarithm problems. After that, GGM has been used for analyzing various assumptions and constructions. Boneh, Boyen, and Goh [11] introduced master theorems for bilinear groups.

Generic bilinear group model $(GBGM)$. There exist three random encoding functions: $\Gamma_1, \Gamma_2, \Gamma_T$ of the additive group $\mathbb{F}_p$, which are injective maps $\Gamma_1, \Gamma_2, \Gamma_T$: $\mathbb{F} \to \{0,1\}^m$, where $m > 3\log p$. For $i = 1, 2, T$, $\mathbb{G}_i$ can be expressed as $\Gamma_i(x) : x \in \mathbb{F}_p$. We are given oracles to compute the induced group action on $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$, and an oracle to compute a non-degenerate bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. Additionally, we are given a random oracle for the hash function $\mathcal{H}$.

Theorem 3. In GBGM, for any adversary $\mathcal{A}$, let q be a bound on the total number of group elements it receives from its queries to the oracles for the hash function, groups $\mathbb{G}_1, \mathbb{G}_2$, and $\mathbb{G}_T$, as well as the bilinear map e, and its communication with the $\mathsf{cc\text{-}Enc_{ABE}}$ security game. Then we have that the advantage of the adversary in the $\mathsf{cc\text{-}Enc_{ABE}}$ security game is $O(q^2/p)$.

Proof. We begin by noting the following standard observation, which can be derived through a straightforward hybrid argument. It is similar to the security proof in [12]. Informally, if we were to define the security game for our scheme in general, it would be distinguishing between challenge ciphertexts corresponding to two different messages. The ciphertext has a component ct_1 which is randomly either $g^{\alpha s} g^{m_0}$ or $g^{\alpha s} g^{m_1}$. Instead, we can consider a modified game in which ct_1 is either $g^{\alpha s}$ or g^{θ}, where θ is uniformly at random from $\mathbb{F}_p$. Considering two hybrids: 1) $g^{\alpha s} g^{m_0}$ and $g^{\theta} g^{m_0}$, 2) $g^{\alpha s} g^{m_1}$ and $g^{\theta} g^{m_1}$, it is straightforward to observe that any adversary with advantage ϵ in the CP-ABE game can be reduced to an adversary with at least $\epsilon/2$ in the modified CP-ABE game. So,

```
$\mathbf{\Pi}_{\mathsf{ccABE}}.\mathsf{Setup}(1^\lambda, \mathsf{ATree}) \to (\mathsf{pp}, \mathsf{mk})$
$\alpha, \beta, \gamma \leftarrow\$ \mathbb{Z}_p$
$(g, h) \leftarrow\$ \mathbb{G}_1 \times \mathbb{G}_2$
Compute a hash table HT
$\mathsf{pp} := (g, h, g^\beta, g^\alpha, g^\gamma, h^{1/\beta}, \mathsf{HT})$
$\mathsf{mk} := (\beta, h^\alpha)$
Return (pp, mk)

$\mathbf{\Pi}_{\mathsf{ccABE}}.\mathsf{Enc}(\mathsf{pp}, \mathsf{pk}, m) \to (\mathsf{ct}; s)$
parse pp as $(g, h, g_1, g_2, g_3, g_4, \mathsf{HT})$
parse pk as ATree
$s \leftarrow\$ \mathbb{Z}_p$
$\mathsf{ct} := \begin{pmatrix} \mathsf{ATree}, g_1^s, g_2^s \cdot g_3^m \\ \left\{ g^{p_x(0)}, H_1(\mathsf{attr}(x))^{p_x(0)} \right\}_{x \in \mathcal{L}} \end{pmatrix}$
Return ct

$\mathbf{\Pi}_{\mathsf{ccABE}}.\mathsf{KeyGen}(\mathsf{pp}, \mathsf{mk}, \mathcal{S}) \to (\mathsf{ck}, \mathsf{pk}, \mathsf{sk})$
parse pp as $(g, h, g_1, g_2, g_3, g_4, \mathsf{HT})$, parse mk as $(\beta, h^\alpha)$
$r, \boldsymbol{t} \leftarrow\$ \mathbb{F} \times \mathbb{F}^{|\mathcal{S}|}$
$\mathsf{ck} := (g_1, g_2, g_3, \{H_1(\mathsf{attr}(x))\}_{x \in \mathcal{L}})$
$\mathsf{pk} := \mathsf{ATree}$    $\mathsf{sk} := \begin{pmatrix} h^{(\alpha + r)\beta^{-1}}, \\ \{h^r \cdot H_2(i)^{t_i}, h^{t_i}\}_{i \in \mathcal{S}} \end{pmatrix}$
Return (ck, pk, sk)

$\mathbf{\Pi}_{\mathsf{ccABE}}.\mathsf{Dec}(\mathsf{pp}, \mathsf{ct}, \mathsf{sk}) \to m$
parse pp as $(g, h, g_1, g_2, g_3, g_4, \mathsf{HT})$
parse ct as $\begin{pmatrix} \mathsf{ATree}, \mathsf{ct}_1, \mathsf{ct}_2, \\ \left\{ \mathsf{ct}_{3,x}, \mathsf{ct}'_{3,x} \right\}_{x \in \mathcal{L}} \end{pmatrix}$, parse sk as $(\mathsf{sk}_1, \left\{ \mathsf{sk}_{2,i}, \mathsf{sk}'_{2,i} \right\}_{i \in \mathcal{S}})$
$O_{\mathsf{rt}} = e(g, h)^{rs} \leftarrow \mathsf{DecNode}(\mathsf{ct}, \mathsf{sk}, \mathsf{rt})$
/ Compute $O_{\mathsf{rt}}$ similar to the original DecNode except for $\frac{e(\mathsf{ct}_{3,x}, \mathsf{sk}_{2,i})}{e(\mathsf{ct}'_{3,x}, \mathsf{sk}'_{2,i})}$
$e(\mathsf{ct}_2, h) \cdot \frac{O_{\mathsf{rt}}}{e(\mathsf{ct}_1, \mathsf{sk}_1)} = e(g_3, h)^m$ / Compute a discrete log to obtain $m$
Return $m$

$\mathbf{\Pi}_{\mathsf{ccABE}}.\mathsf{VerEnc}(\mathsf{ck}, \mathsf{ct}, (s, \mathcal{D}), m) \to \mathsf{true}/\mathsf{false}$
parse ck as $(g_1, g_2, g_3, \{\mathsf{ck}_x\}_{x \in \mathcal{L}})$
parse ct as $\begin{pmatrix} \mathsf{ATree}, \mathsf{ct}_1, \mathsf{ct}_2, \\ \left\{ \mathsf{ct}_{3,x}, \mathsf{ct}'_{3,x} \right\}_{x \in \mathcal{L}} \end{pmatrix}$
parse $\mathcal{D}$ as $(d_{\mathsf{leaf}}, \ldots, d_{\mathsf{rt}})$   / $\mathcal{D}$ denotes the points chosen in $\mathbf{\Pi}_{\mathsf{ccABE}}.\mathsf{Enc}$
return $\mathsf{ct}_1 \overset{?}{=} g_1^s \wedge \mathsf{ct}_2 \overset{?}{=} g_2^s \cdot g_3^m$
        $\wedge \left\{ \mathsf{ct}_{3,x} \overset{?}{=} g^{p_x(0)}, \mathsf{ct}'_{3,x} \overset{?}{=} \mathsf{ck}_x^{p_x(0)} \right\}_{x \in \mathcal{L}}$
```

Figure 3: Our $\mathsf{cc\text{-}Enc_{ABE}}$ based on [12]

we aim to explain the adversary's advantage using the modified game. Now we describe the simulation of the modified CP-ABE game. Let Sim be a simulator for CP-ABE game and $\mathcal{H}_i$ be a hash function for $i = \{1, 2\}$.

Setup phase. Sim chooses α, β at random from $\mathbb{F}_p$. If $\beta = 0$, then Sim aborts. However, the probability of this event occurring is $1/p$. Sim constructs public parameter pp as $(g, h, g_1 = g^\beta, g_2 = g^\alpha, h^{1/\beta})$, and then sends pp to the adversary.

Hash query phase. When the adversary requests the evaluation of $\mathcal{H}$ on a specific string attr_j, a random value a_j is uniformly chosen from $\mathbb{F}_p$. Sim then responds to the queries $\mathcal{H}_1(\mathsf{attr}_j)$ and $\mathcal{H}_2(\mathsf{attr}_j)$ by providing g^{a_j} and h^{a_j} respectively.

Key query phase. Upon receiving the i-th key generation query from the adversary for the attribute set $\mathcal{S}_i$, Sim generates a random r_i uniformly chosen from $\mathbb{F}_p$. For each attribute attr_j in $\mathcal{S}_i$, a random value $r^{i,j}$ is selected from $\mathbb{F}_p$. Subsequently, Sim computes the values as follows: $D = h^{(\alpha+r_i)/\beta}$, $D_j = h^{r_i + a_j r^{i,j}}$, and $D' = h^{r^{i,j}}$. These values are then presented to the adversary.

Challenge phase. Upon receiving a challenge from the adversary consisting of two different messages $m_0, m_1 \in \mathbb{F}_p$, accompanied by the access tree ATree, Sim proceeds as follows. Initially, a random s is chosen from $\mathbb{F}_p$. The linear secret sharing scheme associated with ATree is used to generate shares λ_j of s for each attribute attr_j. It is crucial to emphasize that the λ_j values are uniformly and independently chosen at random from $\mathbb{F}_p$, while adhering to the linear constraints imposed by the secret sharing scheme. Specifically, the λ_j can be simulated by randomly selecting ℓ values $\delta_1, \ldots, \delta_\ell$ from $\mathbb{F}_p$. Then, λ_j can be expressed as a linear combination of δ_k and s. Lastly, Sim chooses a random value θ from $\mathbb{F}_p$. It constructs the encryption components as follows: $\mathsf{ct}_1 = g^\theta$ and $\mathsf{ct}_2 = g_1^s$. For each attribute attr_j, Sim computes $\mathsf{ct}_{3,j} = g^{\lambda_j}$ and $\mathsf{ct}'_{3,j} = g^{a_j \lambda_j}$. These values are then provided to the adversary. It is important to note that if the adversary requests a decryption key for a valid attribute set $\mathcal{S}$ (i.e., it passes the challenge access tree), Sim does not provide the key. Similarly, if the adversary requests a challenge access tree in which one of the previously issued keys passes the access tree, Sim aborts and outputs a random guess.

We prove that the advantage of $\mathcal{A}$ is at most $O(q^2/p)$. The configuration of the $\mathcal{A}$'s oracle query is as follows: 1) $\mathcal{A}$ can only query values that it receives from the Sim or intermediate values obtained through the oracle. 2) there are p distinct values in the space of $\Gamma_1, \Gamma_2, \Gamma_T$. The oracle query can be represented as a rational function f composed of various variables used in the scheme(i.e., α, β, etc). If the values of two different queries (f, f') are the same, we denote it as a Collision, which occurs only if the non-zero polynomial $f - f'$ evaluates to zero. By the Schwartz-Zippel lemma, the probability of a Collision is $O(1/p)$. Therefore, the probability of any such collision happening is at most $O(q^2/p)$ by a union bound.

In the generic group model, the adversary can only perceive the difference of only $\theta = \alpha s$ if there are two distinct queries f, f' such that $f \neq f'$ but $f|_{\theta=\alpha s} = f'|_{\theta=\alpha s}$. As θ only occurs in the form g^θ or $e(g,h)^\theta$, which belongs to $\mathbb{G}_1$ and $\mathbb{G}_T$, f or f' can only be dependent on θ through additive terms of the form $c\theta$, where c is a constant. Thus, we show that the adversary cannot construct an oracle query for $c\alpha s$.

Case 0. We can consider a query $f - f' + c\theta = c\alpha s$ where c is a non-zero. However, the adversary cannot construct a query for $g^{c\alpha s}$ or $e(g,h)^{c\alpha s}$, leading to a contradiction and proving the theorem. Notice that the possibility of querying $g^{\alpha s}$ over $\mathbb{G}_1$ only exists in this case; we will describe the following cases solely for queries over $\mathbb{G}_T$.

Case 1. Note that i is the index of a secret key query, and j and j' are possible attribute strings. The adversary can only construct a term including αs by pairing βs with $(\alpha + r_i)/\beta$ to obtain the term $\alpha s + s r_i$. Further, the adversary could create a linear combination with constants c, c_{sr_i} for some set S (i.e., $c\alpha s + \sum_{i \in S} c_{sr_i} s r_i$). Since the adversary obtains a query polynomial of the form $c\alpha s$, the adversary must add ancillary linear combinations to remove the terms $\sum_{i\in S} c_{sr_i} s r_i$. Observe that the only other term involving monomials $s r_i$ are obtained by pairing $r_i + a_j r_{i,j}$ with some $\lambda_{j'}$, since the $\lambda_{j'}$ terms are linear combinations of s and the μ_k's. Hence, the adversary can make a query polynomial as follows, where A_i is an attribute string set for i-th key query and aux is an auxiliary term:

$$c\alpha s + \sum_{i\in S}\left(c_{sr_i} s r_i + \sum_{(j,j')\in A_i} c_{\lambda_{j'} r_i + \lambda_{j'} a_j r_{i,j}} (\lambda_{j'} r_i + \lambda_{j'} a_j r_{i,j}) \right) + \mathsf{aux}$$

There are two sub-cases: a) there exists some $i \in S$ such that the set of secret shares $L_i = \{\lambda_{j'} : \exists j : (j,j') \in A_i\}$ do not allow for the rebuilding of s, b) For all $i \in S$, the set of secret shares $L_i = \{\lambda_{j'} : \exists j : (j,j') \in A_i\}$ do allow for the rebuilding of s.

Case 1.a. The adversary's query cannot be of the form $c\alpha s$ since the term $s r_i$ is not removed.

Case 1.b. By the assumption that no requested key should pass the challenge access structure and the properties of the secret sharing scheme, we know that the set $L'_i = \{\lambda_j : j \in \mathcal{S}_i\}$ cannot allow for the rebuilding of s. In other words, one share $\lambda_{j'}$ in L_i should be different such that $\lambda_{j'}$ is linearly independent of L'_i. As a result, the adversary's query remains a term of the form $\lambda_{j'} a_j r^{i,j}$ for some $j \in \mathcal{S}_i$. By removing the $\lambda_{j'} a_j r^{i,j}$ term, the adversary may construct a query polynomial for $c\alpha s$. However, since there is no query for the term $\lambda_{j'} a_j r^{i,j}$, the adversary cannot construct a query polynomial for $c\alpha s$, and hence, the possibility of $c\alpha s$ query is not feasible. □

V. Tangram FRAMEWORK

A. *Link* cc-Enc *to* SNARK$_{\mathsf{cc}}$

A ciphertext ct can be seen as a commitment if it is generated from cc-Enc. However, the prover time still needs to be improved due to proving the commitment within a SNARK circuit. Our idea to reduce proving time uses a *proof-dependent commitment* from commit-carrying SNARK and then makes a bridge between the commitment cm and external ciphertext ct using a linkable proof system. This approach is similar to the works [17, 16, 43, 44].

Given a common reference string $\mathsf{crs} := (\mathsf{ek}, \mathsf{vk})$ generated by $\mathbf{\Pi}_{\mathsf{snark}}$.Setup, there exists a SNARK whose proof computation is generated by linear encoding with an evaluation key ek. This proof, computed with a statement $\boldsymbol{x}$ and a witness $\boldsymbol{w}$ playing roles similar to the message, can be viewed as a

```
$\underline{\mathsf{Tangram.Setup}(1^\lambda) \to (\mathsf{pp}, \mathsf{aux}, \mathsf{ck}, \mathsf{pk}, \mathsf{sk})}$
$(\mathsf{pp}, \mathsf{aux}) \leftarrow \mathbf{\Pi}_{\mathsf{cc\text{-}Enc}}.\mathsf{Setup}(1^\lambda)$
$(\mathsf{ck}, \mathsf{pk}, \mathsf{sk}) \leftarrow \mathbf{\Pi}_{\mathsf{cc\text{-}Enc}}.\mathsf{KeyGen}(\mathsf{pp}, \mathsf{aux})$
$\mathbf{Return}\ (\mathsf{pp}, \mathsf{aux}, \mathsf{ck}, \mathsf{pk}, \mathsf{sk})$

$\underline{\mathsf{Tangram.KeyGen}(\mathsf{ck}, \mathcal{R}) \to (\mathsf{ek}, \mathsf{vk})}$
$(\mathsf{ck}_{\mathsf{cc}}, \mathsf{ek}_{\mathsf{cc}}, \mathsf{vk}_{\mathsf{cc}}) \leftarrow \mathbf{\Pi}_{\mathsf{cc}}.\mathsf{Setup}(\mathcal{R})$
Build $\mathcal{R}^{\mathsf{link}}_{\mathsf{eq}}$ from $(\mathsf{pk}, \mathsf{ck}_{\mathsf{cc}}, \mathcal{D}^{\mathsf{link}}_u, \mathcal{D}^{\mathsf{link}}_r, \mathcal{D}^{\mathsf{link}}_o)$
$(\mathsf{ek}_{\mathsf{link}}, \mathsf{vk}_{\mathsf{link}}) \leftarrow \mathbf{\Pi}_{\mathsf{LS}}.\mathsf{KeyGen}(\mathsf{ck}, \mathcal{R}^{\mathsf{link}}_{\mathsf{eq}})$
$\mathbf{Return}\ ((\mathsf{ck}_{\mathsf{cc}}, \mathsf{ek}_{\mathsf{cc}}, \mathsf{ek}_{\mathsf{link}}), (\mathsf{vk}_{\mathsf{cc}}, \mathsf{vk}_{\mathsf{link}}))$

$\underline{\mathsf{Tangram.Enc}(\mathsf{pp}, \mathsf{pk}, m_i) \to (\mathsf{ct}_i; r_i)}$
$(\mathsf{ct}_i; r_i) \leftarrow \mathbf{\Pi}_{\mathsf{cc\text{-}Enc}}.\mathsf{Enc}(\mathsf{pp}, \mathsf{pk}, m_i)$
$\mathbf{return}\ (\mathsf{ct}_i; r_i)$

$\underline{\mathsf{Tangram.Prove}(\mathsf{ek}, u, (\mathsf{ct}_i)_{i\in[l]}; (m_i)_{i\in[l]}, (r_i)_{i\in[l]}, w) \to \pi}$
$(\pi_{\mathsf{cc}}, \mathsf{cm}_{\mathsf{cc}}; o_{\mathsf{cc}}) \leftarrow \mathbf{\Pi}_{\mathsf{cc}}.\mathsf{Prove}(\mathsf{ek}_{\mathsf{cc}}, u; (m_i)_{i\in[l]}, w)$
$\pi_{\mathsf{link}} \leftarrow \mathbf{\Pi}_{\mathsf{LS}}.\mathsf{Prove}(\mathsf{ek}_{\mathsf{link}}, \mathsf{cm}_{\mathsf{cc}}, (\mathsf{ct}_i)_{i\in[l]}, (m_i)_{i\in[l]}, (r_i)_{i\in[l]}, o_{\mathsf{cc}})$
$\mathbf{Return}\ \pi := (\pi_{\mathsf{cc}}, \pi_{\mathsf{link}}, \mathsf{cm}_{\mathsf{cc}})$

$\underline{\mathsf{Tangram.Verify}(\mathsf{vk}, u, (\mathsf{ct}_i)_{i\in[l]}, \pi) \to \mathsf{true/false}}$
Parse $\pi := (\pi_{\mathsf{cc}}, \pi_{\mathsf{link}}, \mathsf{cm}_{\mathsf{cc}})$
$\mathbf{Return}\ \mathbf{\Pi}_{\mathsf{LS}}.\mathsf{Verify}(\mathsf{vk}_{\mathsf{link}}, \mathsf{cm}_{\mathsf{cc}}, (\mathsf{ct}_i)_{i\in[l]}, \pi_{\mathsf{link}}) \wedge$
        $\mathbf{\Pi}_{\mathsf{cc}}.\mathsf{Verify}(\mathsf{vk}_{\mathsf{cc}}, \pi_{\mathsf{cc}}, u, \mathsf{cm}_{\mathsf{cc}})$
```

Figure 4: Generic construction of Tangram from Linker and $\mathsf{SNARK}_{\mathsf{cc}}$. Setup and Enc parts are an external area, which means an offline phase. The dotted box represents the part responsible for generating cc-Enc ciphertexts, which denotes that repeated operations are possible for multiple messages using the same key.

Pedersen-like commitment in which the proving key ek can be seen as the commitment key ck. The consistency of Pedersen commitment can be rephrased as a statement about the knowledge of linear encoding, which can be expressed in algebraic language. Thus, we now need a proof system that proves that the ciphertext generated through commit-carrying encryption and the commitment from commit-carrying SNARK both use the same message internally. Additionally, if the used proof system efficiently proves the relation of Pedersen commitments, we can achieve our desired goal: *fast prover time*.

Connectivity provider(Linker). We need to define a relation $\mathcal{R}^{\mathsf{link}}_{\mathsf{eq}}$ for linking, similar to that described in Legosnark [16] and Eclipse [44], as follows.

$$\mathcal{R}^{\mathsf{link}}_{\mathsf{eq}} = \left\{ \begin{array}{c} ((\boldsymbol{g}, \boldsymbol{h}, \boldsymbol{G}, \boldsymbol{H}, d, d_1, d_2, l), \\ (C, D_1, \ldots, D_l), \\ (\boldsymbol{m}, \boldsymbol{o}, \boldsymbol{r}_1, \ldots, \boldsymbol{r}_l)) \end{array} : \begin{array}{c} C = \boldsymbol{g}^{\boldsymbol{o}} \boldsymbol{h}^{\boldsymbol{m}}, D_i = \boldsymbol{G}^{\boldsymbol{r}_i} \boldsymbol{H}^{\boldsymbol{m}_i} \\ \boldsymbol{g}, \boldsymbol{h} \in \mathbb{G}^{d_1} \times \mathbb{G}^{ld}, \\ \boldsymbol{G}, \boldsymbol{H} \in \mathbb{G}^{d_2} \times \mathbb{G}^{d}, \\ \boldsymbol{m} = \{\boldsymbol{m}_i\}_{i=1}^{l}, \boldsymbol{m}_i \in \mathbb{Z}_p^d \\ \boldsymbol{o} \in \mathbb{Z}_p^{d_1}, \boldsymbol{r}_i \in \mathbb{Z}_p^{d_2}, \end{array} \right\}$$

On a high level, the objective guaranteed by the relation $\mathcal{R}^{\mathsf{link}}_{\mathsf{eq}}$ is that the entire input messages $\boldsymbol{m}_i$ of externally generated ciphertexts and the large message $\boldsymbol{m}$ of proof-dependent commitment generated via SNARK are correctly identical. To elaborate further, when the extractors $\mathcal{E}_{\mathsf{SNARK_{cc}}}$ and $\mathcal{E}_{\mathsf{cc\text{-}Enc}}$ are provided, both extractors should produce the same knowledge when applied to each commitment and ciphertext.

Many NIZKAoK proof systems can prove that two different Pedersen-like commitments open to the same vector under distinct commitment keys. We leverage them to gain connectivity between a commitment cm and a ciphertext ct since $\mathsf{ct} \leftarrow \mathsf{cc\text{-}Enc.Enc}(m) \approx \mathsf{cm} \leftarrow \mathsf{Ped.Com}(m)$. For legibility, we denote these systems as Linker, which is executed by a prover. Attempting to generalize the format of a Linker appears to be a cumbersome task, given the range of available options. However, to enhance readability, we define a concise description of Linker, denoted as $\mathbf{\Pi}_{\mathsf{LS}}$, as follows.

- $\mathbf{\Pi}_{\mathsf{LS}}.\mathsf{KeyGen}(\mathsf{ck}, \mathcal{R})$: takes a commitment key ck and a relation as inputs, and outputs an evaluation key $\mathsf{ek}_{\mathsf{link}}$ and a verification key $\mathsf{vk}_{\mathsf{link}}$.
- $\mathbf{\Pi}_{\mathsf{LS}}.\mathsf{Prove}(\mathsf{ek}_{\mathsf{link}},\ \mathsf{cm},\ (\widehat{\mathsf{cm}}_i)_{i\in[l]},\ (m_i)_{i\in[l]},\ (\boldsymbol{r}_i)_{i\in[l]},\ (o_i)_{i\in[d_1]})$: takes an evaluation key $\mathsf{ek}_{\mathsf{link}}$, a commitment cm, l commitments $(\widehat{\mathsf{cm}}_i)_{i\in[l]}$, l messages $(m_i)_{i\in[l]}$, l randomnesses $(\boldsymbol{r}_i)_{i\in[l]}$ and openings $(o_i)_{i\in[d_1]}$ as inputs, and outputs π_{link}.
- $\mathbf{\Pi}_{\mathsf{LS}}.\mathsf{Verify}(\mathsf{vk}_{\mathsf{link}},\ \mathsf{cm},\ (\widehat{\mathsf{cm}}_i)_{i\in[l]},\ \pi_{\mathsf{link}})$: takes a verification key $\mathsf{vk}_{\mathsf{link}}$, a commitment cm, l commitments $(\widehat{\mathsf{cm}}_i)_{i\in[l]}$ and π_{link}, and accepts (true) or rejects (false) the proof.

B. *Our construction*

We design an encryption-friendly SNARK framework under Pedersen committed engines using three components: cc-Enc, $\mathsf{SNARK}_{\mathsf{cc}}$, and Linker. This framework is denoted as Tangram. Intuitively, a foundational requirement for constructing our framework is the cc-Enc, which generates a key pair (pk, sk) and a commitment key ck. The generated commitment key ck is utilized as an input to the KeyGen of Tangram, and its properties depend on the chosen cc-Enc scheme. In other words, the feature of Tangram may vary based on the specific cc-Enc used. The remaining two components are designed to operate as follows. In the case of $\mathsf{SNARK}_{\mathsf{cc}}$, one generates a commitment $\mathsf{cm}_{\mathsf{cc}}$ under the same message as used in cc-Enc along with a proof π_{cc}. The other component, Linker, serves to connect the external cc-Enc and $\mathsf{SNARK}_{\mathsf{cc}}$ used in the framework. The overall design of the framework follows a generic construction described in Fig.4.

Theorem 4. If $\mathbf{\Pi}_{\mathsf{cc}}$ is a zk-$\mathsf{SNARK}_{\mathsf{cc}}$ over a relation $\mathcal{R}$, the Linker is a NIZKAoK over the relation $\mathcal{R}^{\mathsf{link}}_{\mathsf{eq}}$ and used PKE is a cc-Enc, then the framework Tangram satisfies correctness, knowledge soundness, and zero-knowledge.

Proof. Correctness. The correctness of Tangram can be derived from the correctness of the two constituent schemes, $\mathbf{\Pi}_{\mathsf{cc}}$ and Linker.

Knowledge soundness. Suppose there is an adversary $\mathcal{A}_{\mathsf{Tangram}}$ against Tangram that achieves non-negligible probability of violating the knowledge soundness of Tangram. Assume that the public key pk is generated honestly. Let $\mathcal{A}_{\mathsf{cc}}$ and

$\mathcal{A}_{\mathsf{Linker}}$ be the adversaries against $\mathsf{SNARK}_{\mathsf{cc}}$ and Linker respectively that simulate $\mathcal{A}_{\mathsf{Tangram}}$'s queries to $\mathsf{SNARK}_{\mathsf{cc}}$ and Linker in the games $\mathcal{G}^{\mathsf{KS}}_{\mathsf{cc}}$ and $\mathcal{G}^{\mathsf{KS}}_{\mathsf{Linker}}$. $\mathcal{A}_{\mathsf{Tangram}}$ runs in the game $\mathcal{G}^{\mathsf{KS}}_{\mathsf{Tangram}}$ and outputs a tuple $(u, (\mathsf{ct}_i)_{i \in [l]}, \pi)$ s.t., $\mathsf{Tangram.Verify}(\mathsf{vk}, u, (\mathsf{ct}_i)_{i \in [l]}, \pi) = 1$. Also, let $\mathcal{E}_{\mathsf{Tangram}}$ be the extractor corresponding to adversary $\mathcal{A}_{\mathsf{Tangram}}$ from $\mathcal{E}_{\mathsf{cc}}$ and $\mathcal{E}_{\mathsf{Linker}}$. The extractor $\mathcal{E}_{\mathsf{Tangram}}$ outputs a tuple $((\mathsf{ct}_i)_{i \in [l]}, (m_i)_{i \in [l]}, (r_i)_{i \in [l]})$. We prove that the following probability is *negligible*.

$$\Pr\left[(x, w) \leftarrow \mathcal{G}^{\mathsf{KS}}_{\mathsf{Tangram}, \mathcal{A}_{\mathsf{Tangram}}, \mathcal{E}_{\mathsf{Tangram}}} : (x, w) \notin \mathcal{R}\right] \tag{3}$$

Recall that $\mathcal{A}_{\mathsf{Tangram}}$'s output is composed of $x := (u, (\mathsf{ct}_i)_{i \in [l]}, \pi)$, where π is a tuple consisting of (π_{cc}, $\mathsf{cm}_{\mathsf{cc}}$; o_{cc}). $\mathcal{E}_{\mathsf{Tangram}}$'s output is of the from $w := ((\mathsf{ct}_i)_{i \in [l]}, (m_i)_{i \in [l]}, (r_i)_{i \in [l]})$. If the probability (3) is not negligible, it is equivalent to saying that either the knowledge soundness of $\mathbf{\Pi}_{\mathsf{cc}}$, the knowledge soundness of Linker, or the IND-CPA of cc-Enc has been broken. Therefore, we can conclude that Tangram is knowledge sound.

Zero-knowledge. Likewise to the correctness, the zero-knowledge of Tangram can be derived from the zero-knowledge of the two constituent schemes ($\mathbf{\Pi}_{\mathsf{cc}}$, Linker) and the hiding property of PKE.

The Linker must satisfy at least NIZKAoK, but if the used Linker has succinctness, then the scheme Tangram can satisfy succinctness based on the linker's performance. Thus, the succinctness of Tangram follows by that of the two scheme $\mathbf{\Pi}_{\mathsf{cc}}$ and Linker.

VI. Evaluation and Implementation

Table I: Comparison of $\mathsf{SNARK}_{\mathsf{cc}}$ instantiations. m denotes the number of total wires, n represents the number of multiplication gates, and a denotes the number of addition gates. In addition, $\mathtt{Exp}$ refers to exponentiation, $\mathtt{P}$ denotes pairing, and n-FFT represents n-size fast Fourier transforms.

	crs	$\mathcal{P}$	$\mathcal{V}$	$\lvert\pi\rvert$
ccGro16	$m{+}2n{+}2\ \mathbb{G}_1$, $n\ \mathbb{G}_2$	$m + 3n\ \mathbb{G}_1\mathtt{Exp}$, $n\ \mathbb{G}_2\mathtt{Exp}$	$3\ \mathtt{P}$	$3\ \mathbb{G}_1,\ \mathbb{G}_2$
ccPlonk	$n{+}a\ \mathbb{G}_1$, $\mathbb{G}_2$	$10(n+a)\ \mathbb{G}_1\mathtt{Exp}$, $\approx 55\ n$-FFT	$2\ \mathtt{P}, 18\ \mathbb{G}_1\mathtt{Exp}$	$10\ \mathbb{G}_1, 7\ \mathbb{F}$

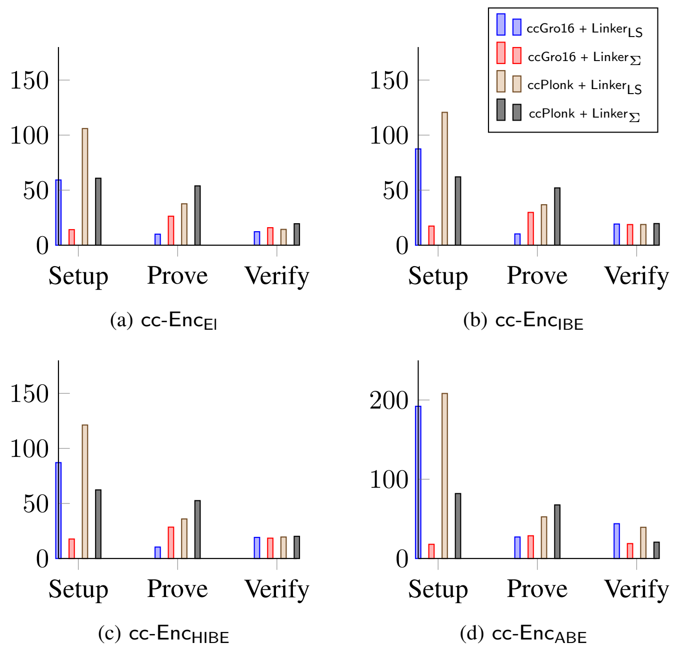
Table II: Comparison of Linker instantiations. We denote by m the number of witness, l the number of input commitments (or ciphertexts), and by d the size of each committed vector

	crs	$\mathcal{P}$	$\mathcal{V}$	$\lvert\pi\rvert$
$\mathsf{Linker}_{\mathsf{LS}}$	$m{+}l{+}1\ \mathbb{G}_1$, $(l+2)\ \mathbb{G}_2$	$m + l\ \mathbb{G}_1\mathtt{Exp}$	$l + 1\ \mathtt{P}$	$1\ \mathbb{G}_1$
Linker_{Σ}	$ld\ \mathbb{G}_1$	$\approx 4\log_2 ld + 6ld\ \mathbb{G}_1\mathtt{Exp}$	$\approx 4\log_2 ld + 2ld\ \mathbb{G}_1\mathtt{Exp}$	$4\lceil\log_2 ld\rceil\ \mathbb{G}_1$, $2\mathbb{F}$

We implement various Tangram frameworks based on top of Arkworks library. Concretely, we leverage the useful libraries, such as finite field and elliptic curve. When encrypting the message on cc-Enc, we split it into chunks of 8 bits and perform encryption on each piece individually. Message length is fixed at 256 bits, and we assume no non-committed instances (i.e., empty ω). Additionally, we adopted the size for the compressed type. Our implemented gadgets for encryption are ccEl, ccIBE, ccHIBE, and ccABE.

$\mathsf{SNARK}_{\mathsf{cc}}$ **instantiations**. We describe the performance for our instantiated $\mathsf{SNARK}_{\mathsf{cc}}$ schemes: ccGro16 [5, 16] and ccPlonk [6]. Table I shows the theoretical costs for our instantiated $\mathsf{SNARK}_{\mathsf{cc}}$. In ccGro16, two additional elements are added to $\mathbb{G}_1$ in crs, and the prover work increases due to $l\ \mathbb{G}_1\mathtt{Exp}$. Further, the proof size increases, but the verifier work decreases accordingly. In ccPlonk, we use a KZG polynomial commitment scheme. Although there is no variation in the public parameter, the prover work incurs an additional cost of $(n + a)\mathbb{G}_1\mathtt{Exp}$ due to generating a commitment. The proof size increases by one element in $\mathbb{G}_1$ and $\mathbb{F}$ due to the aforementioned task.

Linker **instantiations**. As shown in Table II, we illustrate the performance of two Linker instantiations: $\mathsf{Linker}_{\mathsf{LS}}$ [20] and Linker_{Σ} [21, 22, 44]. These schemes are more formally described in the Appendix A. We denote by l the number of input commitments (or ciphertexts) and by d the size of each committed vector. $\mathsf{Linker}_{\mathsf{LS}}$ shows significantly better results compared to Linker_{Σ}; however, $\mathsf{Linker}_{\mathsf{LS}}$ suffers from the issue of trapdoor generation during Setup. Both schemes have verifier work proportional to l, but the operations differ between $\mathsf{Linker}_{\mathsf{LS}}$ and Linker_{Σ}. $\mathsf{Linker}_{\mathsf{LS}}$ involves pairing operations, while Linker_{Σ} performs multiple group scalar multiplications. In terms of proof size, $\mathsf{Linker}_{\mathsf{LS}}$ remains constant, while Linker_{Σ} has a length of $O(\log ld)$.



(a) $\mathsf{cc\text{-}Enc}_{\mathsf{El}}$ (b) $\mathsf{cc\text{-}Enc}_{\mathsf{IBE}}$ (c) $\mathsf{cc\text{-}Enc}_{\mathsf{HIBE}}$ (d) $\mathsf{cc\text{-}Enc}_{\mathsf{ABE}}$

Figure 5: Performance of the execution time across various Tangram frameworks, with the y-axis representing the execution time in milliseconds (ms) for each algorithm.

A. Evaluation

Each experiment was executed on M1 pro with 10 cores and 32GB of RAM, and we performed 100 runs of each experiment and filled in their average. When encrypting the message on cc-Enc, we split it into chunks of `8 bits` and perform encryption on each piece individually. We adopted BLS12-381, a pairing-friendly elliptic curve.

Execution time. We present the performance measured for various Tangram frameworks in figure 5. The execution time is proportional to the size of the message. When splitting a 256-bit message into 8-bit chunks, the proving time shows performance in tens of milliseconds. However, the decryption process, which relies on brute-forcing algorithms, is the main drawback of our scheme and takes the longest time compared to other operations. We observe that performance varies depending on the combination chosen for the intended purpose. For example, if one desires efficient execution time while recognizing the drawback of *trusted setup*, opting for ccGro16 and $\text{Linker}_{\text{LS}}$ is reasonable. On the contrary, if the goal is to achieve verifiability for encryption using a universal CRS after running a trusted setup at once, adopting ccPlonk and Linker_{Σ} is a proper choice.

Size. We measure the size of three major parts within the Tangram framework: public parameter (pp), proof (π), and ciphertext (ct) in Table III. pp can be divided into three main parts: encryption (cc-Enc), SNARK (SNARK_{cc}), and linker (Linker). The size of pp is determined by combining the sizes of all three parts. Similarly, the proof size is obtained by adding the proof sizes of the components SNARK_{cc} and Linker. In the case of ct, general encryption schemes encrypt one message at a time. However, in our scheme, since we only use the short message space, multiple split ciphertexts are generated as output, resulting in a larger size.

Table III: Measurement of the sizes: public parameter, proof, and ciphertext

Microbenchmarks					
SNARK_{cc}	Linker	cc-Enc	|pp|(KB)	$\|\pi\|$(KB)	|ct|(KB)
ccGro16	$\text{Linker}_{\text{LS}}$	El	24.8	0.3	3.3
		IBE	64		4.6
		HIBE	32.9		4.6
		ABE	48.1		13.6
	Linker_{Σ}	El	19.6	1.3	3.3
		IBE	56.8	1.4	4.6
		HIBE	24.6	1.4	4.6
		ABE	20.1	1.5	13.6
ccPlonk	$\text{Linker}_{\text{LS}}$	El	292.3	1.2	3.3
		IBE	332.5		4.6
		HIBE	300.4		4.6
		ABE	315.6		13.6
	Linker_{Σ}	El	287.1	2.3	3.3
		IBE	324.2	2.4	4.6
		HIBE	292.1	2.4	4.6
		ABE	287.6	2.5	13.6

Comparisons. We evaluate the performance for the conventional approach (i.e., *encrypt-in-the-circuit*) to compare the performance with Tangram. [3] We measured the performance

[3]For schemes that originally used type-1 pairings, we applied modifications to switch to type-3 pairings.

Table IV: The performance of Gro16 for varying encryption schemes

Gro16	Setup(s)	Prove(s)	Verify(ms)	|pp|(MB)
ElGamal	0.14	0.12	1.3	1.5
$\text{IBE}_{\text{Water}}$	37.13	35.4	7.88	359.6
HIBE_{BBG}	7.81	7.25	6.31	57.8
ABE_{BSW}	6.382	5.823	6.382	53.4

of the additively-homomorphic ElGamal encryption scheme on the `JubJub` curve, which is a twisted Edwards curve built over the BLS12-381 scalar field. However, for advanced encryption schemes such as IBE, HIBE, and ABE, we choose the BLS12-377 curve for operations such as pairing. As a result, the outer curve used in SNARK is the BW6-761 curve, which has a scalar field equal to the base field of BLS12-377. Performance measurements are conducted based on the BW6-761 curve, and the comparative experiments are specifically carried out for Gro16. For each encryption scheme, we configured the IBE to use the same scalar field bit length as the length of the id. In the case of HIBE, we set the depth d to 32. We set up the system with a policy requiring three attributes for attribute-based encryption and conducted experiments based on this configuration. Although the Tangram framework has some drawbacks regarding proof and ciphertext size and the decryption process compared to traditional methods, it shows exceptional efficiency in setup, proving time, and public parameter size. In fact, for a 256-bit message, the prover time in Tangram is approximately 12x (10.02ms vs. 120ms) - 3500x (10.24ms vs. 35.4s) times faster than the orthodox approach.

B. Implementation

Our framework enables the prover to efficiently prove the knowledge of encryption using zk-SNARKs, making it applicable in scenarios such as digital currency and identity and access management. We experimentally implement our scheme to be plugged into the blockchain. Assume that the encryption scheme can be switched to the cc-Enc and each transaction includes the ciphertext and Tangram proof. Transparency and traceability are ensured through the blockchain's characteristic, while privacy is maintained through the encryption.

Simple scenario. In a blockchain-based system, our framework can be performed as follows: transactions involving the system by individuals encrypt the transaction amount with the central bank's public key, utilizing a commit-carrying encryption. To prove the correctness of the encryption, the user generates a Tangram proof for the generated ciphertext. Subsequently, the user broadcasts a transaction, comprising the ciphertext and proof, to the blockchain-based system. Smart contracts within this system conduct verification of the Tangram proof. Upon successful verification, the transaction is recorded on the blockchain, ensuring the integrity of the transaction.

Contract. In the above scenario, we evaluate the gas consumption performance of the smart contract using the Tangram framework. When constructing a contract, it is imperative

Table V: Gas consumption of $\mathsf{cc\text{-}Enc_{EI}}$

Function	Gas Cost
Deploy	6,519,891
ccGro16.Verify	218,457
Linker.Verify	3,219,491
Total	3,437,948

to incorporate a verifying key within the constructor for the purpose of verifying a Tangram proof. The performance of this verification varies depending on the specific $\mathsf{SNARK_{cc}}$ and Linker. We measure the performance in the context of $\mathsf{cc\text{-}Enc_{EI}}$, ccGro16 and $\mathsf{Linker_{LS}}$. We adopted the BN254 curve, and the contract is deployed on the Ethereum testnet.

Gas consumption. The gas consumption depends on the size of the message blocks because it is closely associated with the verification performance, such as the number of pairing and the size of the verifying key. For 32-block ciphertext, the deployment consumes 6,519,891 gas. The verification cost on ccGro16 is approximately 218,457 gas, while for $\mathsf{Linker_{LS}}$, it rises to about 3,219,491 gas. In the verification process of $\mathsf{Linker_{LS}}$, the operations become increasingly costly due to the $O(l)$ pairing operations, which scale with the number of ciphertexts.

VII. Conclusion

We propose Tangram, an encryption-friendly SNARK framework under Pedersen committed engines, and have designed various commit-carrying encryptions for use in the framework. With our framework, proof generation can be performed significantly faster than the folklore SNARK approach for encryption. Additionally, due to its versatility in achieving composition results of desired features, it can be valuable for a wide range of applications.

References

[1] Y. Chen, X. Ma, C. Tang, and M. H. Au, "Pgc: Decentralized confidential payment system with auditability," in Computer Security – ESORICS 2020, 2020, pp. 591–610.

[2] G. Jeong, N. Lee, J. Kim, and H. Oh, "Azeroth: Auditable zero-knowledge transactions in smart contracts," IEEE Access, vol. 11, pp. 56 463–56 480, 2023.

[3] K. Wüst, K. Kostiainen, N. Delius, and S. Capkun, "Platypus: A central bank digital currency with unlinkable transactions and privacy-preserving regulation," in Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, ser. CCS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 2947–2960. [Online]. Available: https://doi.org/10.1145/3548606.3560617

[4] A. Kiayias, M. Kohlweiss, and A. Sarencheh, "Peredi: Privacy-enhanced, regulated and distributed central bank digital currencies," in Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, ser. CCS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 1739–1752. [Online]. Available: https://doi.org/10.1145/3548606.3560707

[5] J. 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, 2016, pp. 305–326.

[6] A. Gabizon, Z. J. Williamson, and O.-M. Ciobotaru, "Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge," IACR Cryptol. ePrint Arch., vol. 2019, p. 953, 2019.

[7] S. T. V. Setty, "Spartan: Efficient and general-purpose zksnarks without trusted setup," in Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part III, ser. Lecture Notes in Computer Science, D. Micciancio and T. Ristenpart, Eds., vol. 12172. Springer, 2020, pp. 704–737. [Online]. Available: https://doi.org/10.1007/978-3-030-56877-1_25

[8] A. Chiesa, Y. Hu, M. Maller, P. Mishra, N. Vesely, and N. Ward, "Marlin: Preprocessing zksnarks with universal and updatable srs," in Advances in Cryptology – EUROCRYPT 2020, A. Canteaut and Y. Ishai, Eds. Cham: Springer International Publishing, 2020, pp. 738–768.

[9] M. Maller, S. Bowe, M. Kohlweiss, and S. Meiklejohn, "Sonic: Zero-knowledge snarks from linear-size universal and updatable structured reference strings," ser. CCS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 2111–2128. [Online]. Available: https://doi.org/10.1145/3319535.3339817

[10] D. Boneh and X. Boyen, "Efficient selective-id secure identity-based encryption without random oracles," in Advances in Cryptology - EUROCRYPT 2004, C. Cachin and J. L. Camenisch, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 223–238.

[11] D. Boneh, X. Boyen, and E.-J. Goh, "Hierarchical identity based encryption with constant size ciphertext," in Advances in Cryptology – EUROCRYPT 2005, R. Cramer, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 440–456.

[12] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in 2007 IEEE Symposium on Security and Privacy (SP '07). Berkeley, CA, USA: IEEE, 2007, pp. 321–334.

[13] C. Costello, C. Fournet, J. Howell, M. Kohlweiss, B. Kreuter, M. Naehrig, B. Parno, and S. Zahur, "Geppetto: Versatile verifiable computation," in 2015 IEEE Symposium on Security and Privacy. IEEE, 2015, pp. 253–270.

[14] H. Lipmaa, "Prover-efficient commit-and-prove zero-knowledge snarks," in Progress in Cryptology–AFRICACRYPT 2016: 8th International Conference on Cryptology in Africa, Fes, Morocco, April 13-15, 2016, Proceedings 8. Springer, 2016, pp. 185–206.

[15] S. Agrawal, C. Ganesh, and P. Mohassel, "Non-interactive zero-knowledge proofs for composite statements," in Advances in Cryptology – CRYPTO 2018, H. Shacham and A. Boldyreva, Eds. Cham: Springer International Publishing, 2018, pp. 643–673.

[16] M. Campanelli, D. Fiore, and A. Querol, "Legosnark: Modular design and composition of succinct zero-knowledge proofs," Cryptology ePrint Archive, Report 2019/142, 2019, https://eprint.iacr.org/2019/142.

[17] J. Lee, J. Choi, J. Kim, and H. Oh, "SAVER: snark-friendly, additively-homomorphic, and verifiable encryption and decryption with rerandomization," IACR Cryptol. ePrint Arch., p. 1270, 2019. [Online]. Available: https://eprint.iacr.org/2019/1270

[18] B. Waters, "Efficient identity-based encryption without random oracles," in Proceedings of the 24th Annual International Conference on Theory and Applications of Cryptographic Techniques, ser. EUROCRYPT'05. Berlin, Heidelberg: Springer-Verlag, 2005, p. 114–127.

[19] A. Kate, G. M. Zaverucha, and I. Goldberg, "Constant-size commitments to polynomials and their applications," in Advances in Cryptology - ASIACRYPT 2010, M. Abe, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 177–194.

[20] E. Kiltz and H. Wee, "Quasi-adaptive nizk for linear subspaces revisited," in Advances in Cryptology - EUROCRYPT 2015, E. Oswald and M. Fischlin, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 101–128.

[21] T. Attema and R. Cramer, "Compressed Σ-protocol theory and practical application to plug & play secure algorithmics," in Advances in Cryptology – CRYPTO 2020, D. Micciancio and T. Ristenpart, Eds. Cham: Springer International Publishing, 2020, pp. 513–543.

[22] T. Attema, R. Cramer, and S. Fehr, "Compressing proofs of k-out-of-n partial knowledge," in Advances in Cryptology – CRYPTO 2021, T. Malkin and C. Peikert, Eds. Cham: Springer International Publishing, 2021, pp. 65–91.

[23] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof systems," SIAM Journal on Computing, vol. 18, no. 1, pp. 186–208, 1989. [Online]. Available: https://doi.org/10.1137/0218012

[24] B. Parno, J. Howell, C. Gentry, and M. Raykova, "Pinocchio: Nearly practical verifiable computation," in 2013 IEEE Symposium on Security and Privacy. IEEE, 2013, pp. 238–252.

[25] J. Groth, M. Kohlweiss, M. Maller, S. Meiklejohn, and I. Miers, "Updatable and universal common reference strings with applications to zk-snarks," in Annual International Cryptology Conference. Springer, 2018, pp. 698–728.

[26] A. Kosba, D. Papadopoulos, C. Papamanthou, and D. Song, "Mirage: Succinct arguments for randomized algorithms with applications to universal zk-snarks," in Proceedings of the 29th USENIX Conference on Security Symposium, ser. SEC'20. USA: USENIX Association, 2020.

[27] Y. Zhang, D. Genkin, J. Katz, D. Papadopoulos, and C. Papamanthou, "vsql: Verifying arbitrary SQL queries over dynamic outsourced databases," in 2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017. IEEE Computer Society, 2017, pp. 863–880. [Online]. Available: https://doi.org/10.1109/SP.2017.43

[28] T. Xie, J. Zhang, Y. Zhang, C. Papamanthou, and D. Song, "Libra: Succinct zero-knowledge proofs with optimal prover computation," in Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part III, ser. Lecture Notes in Computer Science, A. Boldyreva and D. Micciancio, Eds., vol. 11694. Springer, 2019, pp. 733–764. [Online]. Available: https://doi.org/10.1007/978-3-030-26954-8_24

[29] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, "Bulletproofs: Short proofs for confidential transactions and more," in 2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA. IEEE Computer Society, 2018, pp. 315–334. [Online]. Available: https://doi.org/10.1109/SP.2018.00020

[30] B. Bünz, B. Fisch, and A. Szepieniec, "Transparent snarks from DARK compilers," in Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part I, ser. Lecture Notes in Computer Science, A. Canteaut and Y. Ishai, Eds., vol. 12105. Springer, 2020, pp. 677–706. [Online]. Available: https://doi.org/10.1007/978-3-030-45721-1_24

[31] S. T. V. Setty and J. Lee, "Quarks: Quadruple-efficient transparent zksnarks," IACR Cryptol. ePrint Arch., p. 1275, 2020. [Online]. Available: https://eprint.iacr.org/2020/1275

[32] E. Ben-Sasson, A. Chiesa, M. Riabzev, N. Spooner, M. Virza, and N. P. Ward, "Aurora: Transparent succinct arguments for R1CS," in Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part I, ser. Lecture Notes in Computer Science, Y. Ishai and V. Rijmen, Eds., vol. 11476. Springer, 2019, pp. 103–128. [Online]. Available: https://doi.org/10.1007/978-3-030-17653-2_4

[33] A. Chiesa, D. Ojha, and N. Spooner, "Fractal: Post-quantum and transparent recursive proofs from holography," in Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part I, ser. Lecture Notes in Computer Science, A. Canteaut and Y. Ishai, Eds., vol. 12105. Springer, 2020, pp. 769–793. [Online]. Available: https://doi.org/10.1007/978-3-030-45721-1_27

[34] J. Zhang, T. Xie, Y. Zhang, and D. Song, "Transparent polynomial delegation and its applications to zero knowledge proof," in 2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020. IEEE, 2020, pp. 859–876. [Online]. Available: https://doi.org/10.1109/SP40000.2020.00052

[35] I. Giacomelli, J. Madsen, and C. Orlandi, "Zkboo: Faster zero-knowledge for boolean circuits," in 25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016, T. Holz and S. Savage, Eds. USENIX Association, 2016, pp. 1069–1083. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/giacomelli

[36] J. Katz, V. Kolesnikov, and X. Wang, "Improved non-interactive zero knowledge with applications to post-quantum signatures," in Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, ser. CCS '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 525–537. [Online]. Available: https://doi.org/10.1145/3243734.3243805

[37] M. F. Esgin, R. Steinfeld, J. K. Liu, and D. Liu, "Lattice-based zero-knowledge proofs: New techniques for shorter and faster constructions and applications," in Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part I, ser. Lecture Notes in Computer Science, A. Boldyreva and D. Micciancio, Eds., vol. 11692. Springer, 2019, pp. 115–146. [Online]. Available: https://doi.org/10.1007/978-3-030-26948-7_5

[38] Y. Ishai, H. Su, and D. J. Wu, "Shorter and faster post-quantum designated-verifier zksnarks from lattices," in CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021, Y. Kim, J. Kim, G. Vigna, and E. Shi, Eds. ACM, 2021, pp. 212–234. [Online]. Available: https://doi.org/10.1145/3460120.3484572

[39] J. Groth, "Short pairing-based non-interactive zero-knowledge arguments." in Asiacrypt, vol. 6477. Springer, 2010, pp. 321–340.

[40] H. Lipmaa, "Prover-efficient commit-and-prove zero-knowledge snarks," in Progress in Cryptology–AFRICACRYPT 2016: 8th International Conference on Cryptology in Africa, Fes, Morocco, April 13-15, 2016, Proceedings 8. Springer, 2016, pp. 185–206.

[41] M. Veeningen, "Pinocchio-based adaptive zk-snarks and secure/correct adaptive function evaluation," in Progress in Cryptology-AFRICACRYPT 2017: 9th International Conference on Cryptology in Africa, Dakar, Senegal, May 24-26, 2017, Proceedings 9. Springer, 2017, pp. 21–39.

[42] R. S. Wahby, I. Tzialla, A. Shelat, J. Thaler, and M. Walfish, "Doubly-efficient zksnarks without trusted setup," in 2018 IEEE Symposium on Security and Privacy (SP). IEEE, 2018, pp. 926–943.

[43] M. Campanelli, A. Faonio, D. Fiore, A. Querol, and H. Rodríguez, "Lunar: a toolbox for more efficient universal and updatable zksnarks and commit-and-prove extensions," in Advances in Cryptology–ASIACRYPT 2021: 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6–10, 2021, Proceedings, Part III 27. Springer, 2021, pp. 3–33.

[44] D. F. Aranha, E. M. Bennedsen, M. Campanelli, C. Ganesh, C. Orlandi, and A. Takahashi, "Eclipse: enhanced compiling method for pedersen-committed zksnark engines," in IACR International Conference on Public-Key Cryptography, 2022, pp. 584–614.

[45] J. Nick, T. Ruffing, Y. Seurin, and P. Wuille, "Musig-dn: Schnorr multi-signatures with verifiably deterministic nonces," in Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, 2020, pp. 1717–1731.

[46] Y. Gertner and A. Herzberg, "Committing encryption and publicly-verifiable signcryption," Cryptology ePrint Archive, 2003.

[47] A. Takahashi and G. Zaverucha, "Verifiable encryption from mpc-in-the-head," Cryptology ePrint Archive, Paper 2021/1704, 2021, https://eprint.iacr.org/2021/1704. [Online]. Available: https://eprint.iacr.org/2021/1704

[48] D. Pointcheval, O. Sanders, and J. Traoré, "Cut down the tree to achieve constant complexity in divisible e-cash," in IACR International Workshop on Public Key Cryptography. Springer, 2017, pp. 61–90.

[49] L. Ducas, "Anonymity from asymmetry: New constructions for anonymous hibe," in Topics in Cryptology-CT-RSA 2010: The Cryptographers' Track at the RSA Conference 2010, San Francisco, CA, USA, March 1-5, 2010. Proceedings. Springer, 2010, pp. 148–164.

[50] A. Lewko and B. Waters, "New techniques for dual system encryption and fully secure hibe with short ciphertexts," in Theory of Cryptography, D. Micciancio, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 455–479.

Appendix A
Instantiations of Linker

A. *Linear Subspace SNARK* ($\mathsf{Linker}_{\mathsf{LS}}$)

Briefly, $\mathcal{R}_{\mathsf{eq}}^{\mathsf{link}}$ can also be expressed as the linear subspace relation $\mathcal{R}_{\mathsf{Line}}(\boldsymbol{M}, \boldsymbol{x}, \boldsymbol{w})$ for a committed vector $\boldsymbol{M} \in \mathbb{G}^{(l+1)\times t}$, a public vector $[\boldsymbol{x}]_1 \in \mathbb{G}_1^{l+1}$, and a witness vector $\boldsymbol{w} \in \mathbb{Z}_q^t$ such that:

$$\mathcal{R}_{\mathsf{Line}}(\boldsymbol{M}, \boldsymbol{x}, \boldsymbol{w}) \iff [\boldsymbol{x}]_1 = [\boldsymbol{M}]_1 \cdot \boldsymbol{w}, \text{ where } \boldsymbol{w} \in \mathbb{Z}_q^t$$

Namely, given a fixed public matrix $\boldsymbol{M}$, and a public vector $\boldsymbol{x}$, one can prove knowledge of a vector $\boldsymbol{w}$.

$$\underbrace{\begin{bmatrix} D_1 \\ \vdots \\ D_l \\ C \end{bmatrix}}_{[\boldsymbol{x}]_1 \in \mathbb{G}_1^{l+1}} = \underbrace{\begin{bmatrix} \boldsymbol{G} & 0 & \cdots & 0 & 0 & \boldsymbol{H} & 0 & \cdots & 0 \\ 0 & \boldsymbol{G} & \cdots & 0 & 0 & 0 & \boldsymbol{H} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \boldsymbol{G} & 0 & 0 & 0 & \cdots & \boldsymbol{H} \\ 0 & 0 & \cdots & 0 & \boldsymbol{g} & \boldsymbol{h}_1 & \boldsymbol{h}_2 & \cdots & \boldsymbol{h}_l \end{bmatrix}}_{[\boldsymbol{M}]_1 \in \mathbb{G}_1^{(l+1)\times t}} \cdot \underbrace{\begin{bmatrix} \boldsymbol{r}_1 \\ \vdots \\ \boldsymbol{r}_l \\ \boldsymbol{o} \\ \boldsymbol{m}_1 \\ \vdots \\ \boldsymbol{m}_l \end{bmatrix}}_{\boldsymbol{w} \in \mathbb{Z}_q^t}$$

If the ciphertext ct is generated by cc-Enc, then by considering D_i as the ciphertext ct of cc-Enc and C as the proof-dependent commitment cm of $\mathsf{SNARK}_{\mathsf{cc}}$, the connectivity between the ciphertext ct and the commitment cm can be proven by $\mathsf{CP}_{\mathsf{link}}$. For $\mathcal{R}_{\mathsf{eq}}^{\mathsf{link}}$, we can quite straightforward to build $(\boldsymbol{M}, \boldsymbol{x}, \boldsymbol{w})$. Kiltz-Wee QA-NIZK [20] can prove the pre-mentioned relation. In LegoSNARK [16], this system is introduced by $\mathsf{CP}_{\mathsf{link}}$. We simplify $\mathsf{CP}_{\mathsf{link}}$ for legibility; it can be represented as follows.

- $(\mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{CP}_{\mathsf{link}}.\mathsf{KeyGen}(\boldsymbol{M} \in \mathbb{G}_1^{(l+1)\times t}, \mathcal{R}_{\mathsf{eq}}^{\mathsf{link}})$: Sample $\boldsymbol{k} \xleftarrow{\$} \mathbb{Z}_q^{l+1}, a \xleftarrow{\$} \mathbb{Z}_q$, and output $\mathsf{ek} := (\boldsymbol{M}^\top \cdot \boldsymbol{k}) \in \mathbb{G}_1^t$, $\mathsf{vk} := (\mathsf{vk}_1 = [a]_2 \cdot \boldsymbol{k}, \mathsf{vk}_2 = [a]_2) \in \mathbb{G}_2^{l+1} \times \mathbb{G}_2$.
- $\pi \leftarrow \mathsf{CP}_{\mathsf{link}}.\mathsf{Prove}(\mathsf{ek}, \boldsymbol{x} \in \mathbb{G}_1^{l+1}, \boldsymbol{w})$: Output $\pi \in \mathbb{G}_1 \leftarrow \boldsymbol{w}^\top \cdot \mathsf{ek}$.
- $\mathsf{true/false} \leftarrow \mathsf{CP}_{\mathsf{link}}.\mathsf{Verify}(\mathsf{vk}, \boldsymbol{x}, \pi)$: Verify $e(\boldsymbol{x}^\top, \mathsf{vk}_1) \stackrel{?}{=} e(\pi, \mathsf{vk}_2)$.

Hence, we can immediately plug a ciphertext generated from cc-Enc into $\mathsf{CP}_{\mathsf{link}}$.

B. *Compressed Σ-protocol* (Linker_{Σ})

The protocol Fig.6 for proving equality of committed vectors is described in the Eclipse [44]. The protocol can prove amortization of multiple commitment equality for a relation $\mathcal{R}_{\mathsf{AmEq}}$, which is identical to our relation $\mathcal{R}_{\mathsf{eq}}^{\mathsf{link}}$, thereby making our relation easily portable. However, as the proof size is linearly dependent on the response vector $\boldsymbol{z} \in \mathbb{Z}_q^l$. We can reduce the proof size using Attema and Cramer's framework [21] and Attema, Cramer, and Fehr's framework [22]. Intuitively, since the transcript elements $x, A, \hat{A}, c, \boldsymbol{s}$, and $\boldsymbol{u}$ are fixed, it should suffice to prove knowledge of $\boldsymbol{z}$ such that $\boldsymbol{h}^z \stackrel{?}{=} Y := A \cdot C^c \cdot \boldsymbol{g}^{-\boldsymbol{s}}$ and $\hat{\boldsymbol{H}}^{\boldsymbol{z}} \stackrel{?}{=} Y' := \hat{A} \cdot \prod_{i\in[l]}(D_i^{x^i})^c \cdot \boldsymbol{G}^{-\boldsymbol{u}}$ rather than sending $\boldsymbol{z}$ itself. This is where the principles of compressed Σ-protocol theory come into effect. A compressed relation $\mathcal{R}_{\mathsf{AmEq}}^{\mathsf{Com}}$ can be expressed as stated below.

$$\mathcal{R}_{\mathsf{AmEq}}^{\mathsf{Com}} = \left\{ \left(l, (\boldsymbol{h}, \hat{\boldsymbol{H}}), (Y, Y'), \boldsymbol{z} \right) : Y = \boldsymbol{h}^{\boldsymbol{z}}, Y' = \hat{\boldsymbol{H}}^{\boldsymbol{z}} \right\}$$

Protocol. Amortized equality protocol for relation $\mathcal{R}_{\mathsf{eq}}^{\mathsf{link}}$

$\mathcal{P}$ $\qquad\qquad\qquad\qquad\qquad\qquad$ $\mathcal{V}$

$$\xleftarrow{\mathcal{V} \text{ sends randomness } x \in \mathbb{Z}_q}$$

$$\xleftrightarrow{\text{Both Compute } \hat{\boldsymbol{H}} = (\boldsymbol{H}^{x^i})_{i\in[l]}}$$

$\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma} \leftarrow\!\!\$\ \mathbb{Z}_q^{ld\times d_1 \times d_2}$

$$\xrightarrow{\mathcal{P} \text{ sends } A = \boldsymbol{g}^{\boldsymbol{\beta}} \boldsymbol{h}^{\boldsymbol{\alpha}}, \hat{A} = \boldsymbol{G}^{\boldsymbol{\gamma}} \hat{\boldsymbol{H}}^{\boldsymbol{\alpha}}}$$

$$\xleftarrow{\mathcal{V} \text{ sends random challenge } c \in \mathbb{Z}_q}$$

$$\xrightarrow[\boldsymbol{u} = \boldsymbol{\gamma} + c\Sigma_{i\in[l]} r_i x^i]{\boldsymbol{z} = \boldsymbol{\alpha} + c\boldsymbol{m}, \boldsymbol{s} = \boldsymbol{\beta} + c\boldsymbol{o},}$$

Check

$$\boldsymbol{g}^s \cdot \boldsymbol{h}^{\boldsymbol{z}} \stackrel{?}{=} A \cdot C^c$$

$$\hat{\boldsymbol{H}}^{\boldsymbol{z}} \cdot \boldsymbol{G}^{\boldsymbol{u}} \stackrel{?}{=} \hat{A} \cdot \prod_{i\in[l]} (D_i^{x^i})^c$$

Figure 6: Amortized equality protocol for relation $\mathcal{R}_{\mathsf{eq}}^{\mathsf{link}}$

Protocol. Compressed Σ-protocol for relation $\mathcal{R}_{\mathsf{AmEq}}^{\mathsf{Com}}$

$\mathcal{P}$ $\qquad\qquad\qquad\qquad\qquad\qquad$ $\mathcal{V}$

if $l = 1$ **Check**

$$h^z \stackrel{?}{=} Y, \hat{H}^z \stackrel{?}{=} Y'$$

$$\xrightarrow[\hat{\boldsymbol{L}} = \hat{\boldsymbol{H}}_R^{\boldsymbol{z}_L}, \hat{\boldsymbol{R}} = \hat{\boldsymbol{H}}_L^{\boldsymbol{z}_R}]{\boldsymbol{L} = \boldsymbol{h}_R^{\boldsymbol{z}_L}, \boldsymbol{R} = \boldsymbol{h}_L^{\boldsymbol{z}_R},}$$

$$\xleftarrow{\mathcal{V} \text{ sends randomness } \delta \in \mathbb{Z}_q}$$

$\boldsymbol{z} = \boldsymbol{z}_L + \delta \cdot \boldsymbol{z}_R$

$$\xleftrightarrow[\boldsymbol{h} = \boldsymbol{h}_L^\delta \boldsymbol{h}_R, \hat{\boldsymbol{H}} = \hat{\boldsymbol{H}}_L^\delta \hat{\boldsymbol{H}}_R]{Y = \boldsymbol{L} Y^\delta \boldsymbol{R}^{\delta^2}, Y' = \hat{\boldsymbol{L}} (Y')^\delta \hat{\boldsymbol{R}}^{\delta^2}}$$

$$\xleftrightarrow[(l/2, (\boldsymbol{h}, \hat{\boldsymbol{H}}), (Y, Y'), \boldsymbol{z})]{\text{Repeat the protocol for the instance}}$$

Figure 7: Compressed Σ-Protocol for relation $\mathcal{R}_{\mathsf{AmEq}}^{\mathsf{Com}}$

One way to reduce to the desired proof size is to cut the vector in half and merge the cropped vectors using a linear combination. If we treat the combined vector as exponent, we can think of it as committing to the vector with size of $l/2$. Let us crop the vector as $\boldsymbol{h} = (\boldsymbol{h}_L, \boldsymbol{h}_R)$, $\hat{\boldsymbol{H}} = (\hat{\boldsymbol{H}}_L, \hat{\boldsymbol{H}}_R)$, $\boldsymbol{z} = (\boldsymbol{z}_L, \boldsymbol{z}_R)$ with the size of $l/2$. We described the compressed Σ-protocol for $\mathcal{R}_{\mathsf{AmEq}}^{\mathsf{Com}}$ in Fig.7.