Sigma Protocols from Verifiable Secret Sharing and Their Applications

Min Zhang[1], Yu Chen[1(✉)], Chuanzhou Yao[1], and Zhichao Wang[2]

[1] School of Cyber Science and Technology, Shandong University, Qingdao 266237, China
[2] School of Cyber Security, University of Chinese Academy of Sciences, Beijing 100049, China
{zm_min,yaochuanzhou}@mail.sdu.edu.cn, yuchen@sdu.edu.cn, wangzhichao2022@iie.ac.cn

Abstract. Sigma protocols are one of the most common and efficient zero-knowledge proofs (ZKPs). Over the decades, a large number of efficient Sigma protocols are proposed, yet few works pay attention to the common design principal. In this work, we propose a generic framework of Sigma protocols for algebraic statements from verifiable secret sharing (VSS) schemes. Our framework provides a general and unified approach to understanding Sigma protocols for proving knowledge of openings of algebraic commitments. It not only neatly explains the classic protocols such as Schnorr, Guillou–Quisquater and Okamoto protocols, but also leads to new Sigma protocols that were not previously known.

Furthermore, we show an application of our framework in designing ZKPs for composite statements, which contain both algebraic and non-algebraic statements. We give a generic construction of ZKPs for composite statements by combining Sigma protocols from VSS and ZKPs following MPC-in-the-head paradigm seamlessly via a technique of *witness sharing reusing*. Our construction has advantages of requiring no trusted setup, being public-coin and having a fast prover runtime. By instantiating our construction using Ligero++ (Bhadauria et al., CCS 2020), we obtain a new ZK protocol for composite statements, which achieves a new balance between running time and the proof size, thus resolving the open problem left by Backes et al. (PKC 2019). Concretely, the proof size is polylogarithmic to the circuit size and the number of public-key operations that both the prover and the verifier require is independent to the circuit size.

Keywords: Sigma protocols · Verifiable secret sharing · Composite statements · MPC-in-the-head.

Table of Contents

Sigma Protocols from Verifiable Secret Sharing and Their Applications 1
Min Zhang, Yu Chen(✉), Chuanzhou Yao, and Zhichao Wang
1 Introduction 3
1.1 Our Contributions 5
1.2 Related Work 8
2 Preliminaries 9
2.1 Commitment Schemes 9
2.2 Sigma Protocols 10
2.3 Secure Multiparty Computation 10
2.4 (Verifiable) Secret Sharing 11
3 A Framework of Sigma Protocols From VSS 11
3.1 A Refined Definition of VSS Schemes 11
3.2 The Framework of Sigma Protocols 13
4 Instantiations of Our Framework 15
4.1 Proof of Knowledge of A Discrete Logarithm 15
4.2 Proof of Knowledge of Several Discrete Logarithms 15
4.3 Proof of Knowledge of A Representation 17
4.4 Proof of Knowledge of An eth Root 17
5 A Framework of ZKPs for Composite Statements 18
5.1 A Generalization of MPC-in-the-Head Paradigm 18
5.2 Separable VSS Schemes 21
5.3 Generic Construction of ZKPs for Composite Statements 22
6 An Instantiation of ZKP for Composite Statements 24
6.1 Review of Ligero++ 24
6.2 A Sigma Protocol for Pedersen Commitments 25
7 Conclusion 27
A Missing Proofs 30
A.1 A Proof for Theorem 2 30
A.2 A Proof for Theorem 3 30
A.3 A Proof for Theorem 4 31
A.4 A Proof for Theorem 5 31
A.5 A Proof for Theorem 8 32

1 Introduction

Zero-knowledge proofs (ZKPs), introduced by Goldwasser, Micali and Rackoff [GMR85], allow a prover to convince a verifier that a statement is true without revealing any further information. Goldreich, Micali, and Wigderson [GMW86] further showed that ZKP exists for every $\mathcal{NP}$ language, making it an extremely powerful tool in modern cryptography. Since its introduction in the mid 1980s, ZKPs have been used as an instrumental building block in a myriad of cryptographic protocols/schemes like identification protocols [FFS87], digital signatures [BCC+16, Sch91], CCA-secure public-key encryption [NY90, Sah99], anonymous credentials [CL01], voting [CF85], maliciously secure multi-party computation [GMW87], and privacy-preserving cryptocurrency [GK15, BCG+14].

In the realm of ZKPs[1], there are three types of statements. The first is algebraic statements, which are defined by relations over algebraic groups like prime-order groups and RSA-type groups, such as knowledge of discrete logarithm or modular root. The second is non-algebraic statements, which are expressed by arithmetic/boolean circuits, such as knowledge of preimage of SHA256 or knowledge of plaintext of AES encryption. The third is composite statements that mix algebraic and non-algebraic statements, e.g. the value w committed by Com also satisfies $C(w) = 1$, where the predicate C represents an arithmetic/boolean circuit. Below we briefly survey ZKPs for the three types of statements.

ZKPs for non-algebraic statements. Since boolean/arithmetic circuits can describe arbitrary computations, ZKPs for non-algebraic statements are usually referred to as general-purpose. The last decade has seen tremendous progress in designing and implementing efficient general-purpose ZKPs (see [Tha22] for a comprehensive survey). These efforts can be roughly divided into four categories according to the underlying machinery.

The first is built upon probabilistic checkable proof (PCP). Following the seminal works of Kilian [Kil92] and Micali [Mic94] based on classical PCPs, recent works [AHIV17, BBHR18, BCR+19, ZXZS20, COS20, Set20] begin to build general-purpose ZKP from interactive variants of PCP, first in the model of interactive PCP [KR08] and then in the more general model of interactive oracle proofs [BCS16, RRR16]. ZKPs of this category have the advantages of not relying on public-key cryptography, not requiring trusted setup, and offering conjectured post-quantum security. The second is based on linear PCP, initiated by Ishai, Kushilevitz, and Ostrovsky [IKO07], and followed by a sequence of works [Gro10, Lip12, GGPR13, Gro16, MBKM19, CHM+20]. ZKPs of this category are featured with constant size proofs and fast verification, but they are quite slow on the prover side and require long and "toxic" common reference string. The third is based on inner product arguments. Initial work [Gro09] of this line has square root size proof and linear verification time. Followup works [BCC+16, BBB+18] managed to achieve logarithmic size proof, and the verification cost is finally reduced to logarithmic complexity [Lee21]. The fourth is based on garbled circuits. The original protocol due to Jawurek, Kerschbaum and Orlandi [JKO13] is secret-coin in nature. Recently, Cui and Zhang [CZ21] showed how to tweak the JKO protocol to public-coin. ZKPs of this category require linear prover time, proof size and verification time.

ZKPs for algebraic statements. Almost exclusively, the most common and efficient ZKPs for algebraic statements fall into a class known as Sigma protocols, introduced by Cramer [Cra96]. Let L be an $\mathcal{NP}$ language associated with relation R, i.e., $L = \{x \mid \exists w \text{ s.t. } R(x, w) = 1\}$. A Sigma ($\Sigma$) protocol for L is a 3-move public coin interactive proof system that allows a prover to convince a verifier that he knows a witness w of a public instance x without disclosing w. The Greek letter Σ visualizes the 3-move structure (commit, challenge and response). The prover sends an initial message a called a commitment to the verifier, the verifier replies with a uniformly and independently random chosen challenge e from some finite challenge space, and the prover answers with a response z as the final message. Finally, the verifier decides whether to accept or reject the statement based on the transcript (a, e, z).

Sigma protocols are very appealing due to many attractive properties. First, Sigma protocols are extremely efficient for algebraic statements. They yield short proof sizes, only require a constant number of

[1] For the sake of convenience, we will not distinguish between computational and information-theoretic soundness, and thus refer to both proofs and arguments as "proofs".

public-key operations and do not need trusted common reference string generation. Although seemingly specific, Sigma protocols for algebraic statements cover a wide variety of tasks arise from practice such as proving the knowledge of discrete logarithm/modular root, a tuple is of the Diffie-Hellman type, an ElGamal/Paillier encryption is to a certain value, and many more. Second, Sigma protocols are closed under parallel composition, and thus it is possible to efficiently combine several simple Sigma protocols to prove compound statements. This further increases the usability of Sigma protocols. Third, the so-called special soundness make Sigma protocols easy to work with by providing a simple way to establish proof of knowledge property. Moreover, Sigma protocols can be made non-interactive using the Fiat-Shamir heuristic [FS86]. The above properties make Sigma protocols an incredibly powerful tool for various cryptographic tasks.

In contrast to the state of affairs of general-purpose ZKP, though Sigma protocols have been intensively studied in the last four decades, few attentions are paid to generic constructions. This is probably because that the design of Sigma protocols is relatively easier than that of general-purpose ZKPs. Sigma protocols in the literature such as the classic Schnorr [Sch91], Batching Schnorr [GLSY04], Guillou-Quisquater [GQ88], and Okamato protocol [Oka92] are ingenious but hand-crafted, and they came with a separate proof. It is curious to know whether there exists a common design principal.

ZKPs for composite statements. A composite statement is one that contains both algebraic function and non-algebraic function, e.g., x is a Pederden commitment to w such that $\mathrm{SHA256}(w) = y$. As noted in [CGM16, AGM18, BHH+19], ZKPs for composite statements have various applications, such as proof of solvency for Bitcoin exchanges, anonymous credentials based on RSA and (EC-)DSA signatures, and 2PC with authenticated input.

To prove composite statements, a naïve approach is transforming composite statements into a single form, namely either algebraic or non-algebraic form, and using only Sigma protocols or general-purpose ZKPs to prove it. Precisely, for a non-algebraic statement expressed as a circuit, one could express each gate of the circuit as an algebraic relation between input and output. However, such transformation will introduce significant overhead when the circuit is large, since it would cost several exponentiations per gate in the circuit. As noted by [AGM18], in case of hash functions and block-ciphers, it costs tens of thousands of exponentiations and group elements when proving the associated algebraic relations of the circuits. Alternatively, one could turn the algebraic part into non-algebraic structure. But it also results in a substantial increase in proof size, since the circuit for computing a single exponentiation could be of thousands or millions of gates depending on the group size.

A better approach, employed by most of prior works on this direction, is that: using Sigma protocols to prove the algebraic part, using off-the-shelf efficient general-purpose ZKPs to prove the non-algebraic part, then additionally designing customized Sigma protocols as a "glue" to link the two parts. The resulting proof systems will inherent the advantages and disadvantages of the underlying general-purpose ZKPs. These proof systems could be divided based on the general-purpose ZKPs they used. Roughly, [CGM16] presented two tailor-made "glue" proofs to link with the JKO protocol [JKO13] protocol which lies in general-purpose ZKPs of the fourth category aforementioned; the three works [AGM18, CFQ19, ABC+22] each gave a generic construction of "glue" proofs to link with ZKPs based on Linear PCP and pairings; general-purpose ZKPs based on inner product arguments [BBB+18, HKR19] have the advantages of being able to combine with Pedersen-like commitments readily, as their main technique is using the Sigma protocols to prove the associated algebraic relations of the circuits.

However, very few works focused on designing ZKPs for composite statements using general-purpose ZKPs of the first category. Though the advantage of post-quantum security seems less important in the scenarios of composite statements, most of them also enjoy the advantages of having a fast prover runtime, especially those following MPC-in-the-head paradigm, which forms an important subclass of general-purpose ZKPs based on PCP machinery. To our knowledge, [BHH+19] is the only work which uses ZKPs following MPC-in-the-head paradigm. However, their construction is not generic, making their construction difficult to optimize. The "glue" proofs in [BHH+19] are tailored for the ZKBoo [GMO16]/ZKB++ [CDG+17] protocols which have the cons of large proof size, thus not being able to be used to link with other proof systems like Ligero [AHIV17]/Ligero++ [BFH+20]. It is not clear how to give a generic construction of efficient ZKPs for composite statements, using prover efficient general-purpose ZKPs like those based on MPC-in-the-head paradigm.

The above discussion motivates the main questions that we study in this paper:

Is there a generic framework of Sigma protocols? Can this framework help to give a generic construction of prover efficient ZKPs for composite statements?

1.1 Our Contributions

In this work, we positively answer the above two questions and summarize our contributions as below.

1.1.1 A Framework of Sigma Protocols for Algebraic Statements

We present a framework of Sigma protocols for algebraic statements from verifiable secret sharing (VSS) schemes. Our framework not only neatly explains existing classic Sigma protocols including the Schnorr, Batching Schnorr, GQ, and Okamoto protocols, but also provides a unified paradigm of designing Sigma protocols for proving knowledge of openings of algebraic commitments.

MPC-in-the-head paradigm revisit. Ishai et al. [IKOS07] showed how to build general-purpose ZKPs by using MPC in a black-box manner. In a nutshell, their construction proceeds by having the prover simulate an execution of an n-party secure-computation protocol Π_f that evaluates the function $f_y(w_1, \ldots, w_n)$ which outputs "1" iff $C(y, w_1 \oplus \cdots \oplus w_n) = 1$, and commit the views of the parties in the protocol. The verifier then picks a random subset of those parties, and the prover opens the corresponding views. The verifier finally accepts if the opened views all output "1" and are consistent with each other. Their approach, known as the MPC-in-the-head, presents a generic connection between these two fundamental notions, and gives rise to a rich line of general-purpose ZKPs with continually improved performance, including ZKBoo [GMO16], ZKB++ [CDG+17], KKW [KKW18], Ligero [AHIV17], Ligero++ [BFH+20] and more, forming a promising subclass of general-purpose ZKPs from symmetric-key primitives. These protocols are featured with requiring no trusted setup, being able to be made non-interactive via the Fiat-Shamir heuristic [FS86] and having a good prover runtime [Tha22]. Interestingly, we find that the ZKPs from the MPC-in-the-head paradigm also follow the commit-challenge-response pattern. In light of this observation, the MPC-in-the-head paradigm actually gives a generic construction of Sigma protocols for non-algebraic statements, and the power of MPC makes the proof system general-purpose. This suggests that when seeking for a framework of Sigma protocols for algebraic statements, we may start from some lighter machinery than MPC.

VSS-in-the-head. An (n, t_p, t_f)-verifiable secret sharing (VSS) scheme allows a dealer to distribute a secret s among n participants, in such a way that no group of up to t_p participants could learn anything about s and any group of at least t_f participants could recover the secret. It is an essential building block employed in numerous MPC protocols [GMW87, BGW88, CCD88, RB89] to achieve security against malicious players. From this point of view, VSS is arguably the right backbone of Sigma protocols for algebraic statements.

A REFINED DEFINITION OF VSS. In this work, we restrict ourselves to non-interactive VSS schemes. For simplicity, we will omit non-interactive hereafter when the context is here. Before describing the framework, we first give a refined definition of VSS. Compared to the original definition proposed by Feldman [Fel87], there are two primary differences in syntax: (1) The secret is committed rather than being encrypted, such relaxation makes our definition more general; (2) We explicitly require the sharing algorithm additionally outputs *authentication information*, denoted by *aut*, which essentially commits to the sharing method (e.g., in the case of Feldman's VSS scheme, it is a vector of commitments to the polynomial's coefficients), and will later be used to check the validity of each share. This kind of information is omitted in the original definition. Furthermore, we also refined the security properties of VSS: (1) For correctness, we do not stipulate that the secrets recovered by different groups of participants are consistent as in [Fel87], instead we require that the recovered secrets must be an opening of the commitment. This property is crucial for some VSS applications and is actually met by many existing VSS schemes, but it has never been formally defined; (2) For privacy, we provide a simulation-based definition instead of a game-based one, making it more convenient to use in the context of ZKP and MPC. The details of the refined definition and more differences between the two definitions can be found in Section 3.1.

Sigma protocols from VSS. Having settled on a satisfactory definition of VSS, we are ready to describe the framework of Sigma protocols for algebraic statements—"given a commitment x, prove the knowledge of an opening (s, r) such that $\mathsf{Com}(s; r) = x$". Our framework is built upon (n, t_p, t_f)-VSS schemes (where the secret is committed via Com). In the commit phase, the prover acts as a dealer sharing the witness (s, r) into n pieces of shares $v_1, \ldots, v_n$ "in his head" (or simply generating the compact description of the sharing method for efficiency consideration) and generating the associated authentication information aut, then sends aut to the verifier. In the challenge phase, the verifier picks a random subset $I \subseteq [n]$ where $|I| \leq t_p$ (here $[n]$ serves as the challenge space), and acts as the set of participants $\{P_i\}_{i \in I}$ querying for their private shares. In the response phase, the prover answers with corresponding shares $(v_i)_{i \in I}$. Finally, the verifier decides to accept or reject the statement by checking whether each v_i is a valid share for participant P_i. Intuitively, the *special soundness* property follows from the *correctness* of VSS and the *special honest verifier zero-knowledge property* follows from the *privacy* of VSS.

The above framework from VSS encompasses almost all the classic Sigma protocols for proving knowledge of openings of algebraic commitments. As a concrete example, we show how to derive the celebrated Schnnor protocol from our framework. The start point is the Feldman's (n, t_p, t_f)-VSS scheme [Fel87] where $t_f = t_p+1$: to distribute a secret $s \in \mathbb{F}_p$ among n participants $P_1, \ldots, P_n$, the dealer first computes a commitment $c = g^s$ to secret s, chooses a t_p-degree polynomial $f(x) = a_0 + a_1 x + \cdots + a_{t_p} x^{t_p}$ where $a_0, \ldots, a_{t_p-1} \xleftarrow{\text{R}} \mathbb{F}_p$ and $a_{t_p} = s$ (the coefficients of the polynomial could be viewed as the compact description of the sharing method), and sets the private share v_i for P_i as $f(i)$, then generates the commitment of the randomnesses as the authentication information, i.e., $aut = (c_0, \ldots, c_{t_p-1})$ where $c_j = g^{a_j}$ for $0 \leq j \leq t_p - 1$. The dealer then broadcasts c and aut, and sends v_i to P_i in private. Upon receiving the share, each participant checks the validity of the share with respect to c and aut, and rejects if it is invalid. The secret s can be recovered by pooling more than t_p valid shares. By setting the number of participants n to p (the size of the field $\mathbb{F}_p$), the privacy threshold t_p to 1, we immediately recover the classic Schnorr protocol from our framework. More examples can be found in Section 4.

1.1.2 A Framework of ZKPs for Composite Statements

To demonstrate the usefulness of our framework, we show its application in designing ZKPs for composite statements.

As aforementioned, most of the proof systems in this realm require "glue" proofs to prove that the same witness is used to generate proofs for the algebraic part and the non-algebraic part. Without these proofs, a cheating prover can easily generate proofs for the two parts independently using inconsistent witnesses. In [BHH+19], the "glue" proofs are tailor-made for ZKBoo/ZKB++, making the overall construction semi-generic and difficult to optimize. An intriguing question is whether the seemingly indispensable "glue" proofs are necessary when designing ZKPs for composite statements. In this work, we show that by reusing the witness sharing process, Sigma protocols from VSS and ZKPs following MPC-in-the-head paradigm can be combined seamlessly, yielding a generic construction of ZKPs for composite statements. The framework inherits the benefits of MPC-in-the-head paradigm, thus enjoying advantages of having a fast prover. The efficiencies of concrete instantiations depend on the underlying general-purpose ZKPs and the tailored Sigma protocols.

Enforcing consistency via witness sharing reusing. Observe that Sigma protocols from VSS and ZKPs from MPC bear strong resemblance. Both of them follow similar pattern: at the very beginning the provers share the witness in their heads; in the challenge phase the verifiers ask to reveal a subset of witness shares; in the response phase the provers reply with corresponding shares (albeit in ZKPs from MPC, the shares are included as a part of parties' views). Our main insight is that, by reusing the witness sharing procedure, Sigma protocols from VSS and ZKPs from MPC can be seamlessly combined together to prove composite statements without any additional "glue" proofs. More precisely, we prove the algebraic part using Sigma protocols from VSS, and prove the non-algebraic part using ZKPs from MPC. In particular, the overlap parts of witness sharing, challenge and response are reused to reduce costs and enforce (possibly malicious) prover to use identical witness when generating the proofs for algebraic and non-algebraic parts. Since both Sigma protocols from VSS and general-purpose ZKPs can be made non-interactive via the Fiat-Shamir heuristic [FS86], the combination of them can be made non-interactive as well.

Our construction gets rid of "glue" proofs that were used to enforce consistency in prior works. This is because the prover only shares the witness in his head once and distributes the shares to the same group of participants later, whereas the verifier accepts if and only if the shares pass verifications of both algebraic and non-algebraic parts. From the security proof perspective, when formally proving special soundness, we can construct an extractor Ext by invoking extractors Ext_{Σ} of Sigma protocols from VSS and $\mathsf{Ext}_{\mathrm{ZKP}}$ of ZKPs from MPC as subroutines. Since the witness sharing procedure are reused, Ext_{Σ} and $\mathsf{Ext}_{\mathrm{ZKP}}$ in fact run the same recovering algorithm on the same input, and thus output the same witness satisfying both algebraic and non-algebraic statements.

When implementing above high-level idea, we encounter the following technical challenge: the mechanisms used in the original ZKPs from MPC [IKOS07] and the Sigma protocols from VSS are different, causing the two proof systems incompatible. More precisely, the former sticks to the XOR-based secret sharing (SS) schemes (a special case of $(n, n-1, n)$-SS), while the latter stems from (n, t_p, t_f)-VSS schemes. We overcome this obstacle in two steps. First, we generalize the MPC-in-the-head paradigm by replacing the XOR-based SS scheme with (n, t_p, t_f)-SS schemes and rigorously prove its proof of knowledge property, which might be of independent interest. Second, we introduce a mild property called *separability* for VSS which is a mild property satisfied by many existing VSS schemes. Roughly speaking, we say a VSS scheme satisfies *separability* if its produce of generating shares $(v_1, \ldots, v_n)$ and authentication information *aut* could be separated and the shares are generated as per some secret sharing schemes. Such delicate dissection allows us to distill the common secret sharing mechanism used in Sigma protocols from VSS and ZKPs from MPC, paving the way to implement the *witness sharing reusing* technique.

An efficient instantiation. We instantiate above framework of ZKPs for composite statements by choosing Ligero++ [BFH+20] as the underlying general-purpose ZKPs and designing a Sigma protocol from VSS which is with respect to the SS component underlying Ligero++. Since the SS components used by Ligero [AHIV17] and Ligero++ are almost the same, the tailored Sigma protocol could also be combined with Ligero seamlessly by choosing appropriate parameters. This will lead to a faster prover while a larger proof size. The resulting protocol is public-coin, requires no trusted setup and achieves a new trade-off between proof size and running time. Concretely, the proof size is polylogarithmic to the circuit size and the number of expensive public-key operations that both the prover and the verifier require is independent to the circuit size. See Section 6.2 for a detailed efficiency analysis.

Table 1 shows a brief comparison between closely related works. Compared to [BHH+19], this instantiation achieves a asymptotically smaller proof size, thus settling the open problem in [BHH+19]: whether a more compact ZKP for composite statements can be constructed by using Ligero/Ligero++. However, in terms of efficiency (both the prover and the verifier side), our instantiation is evidently inferior to the protocols in [BHH+19]. Actually, it is hard to give a more prover efficient ZKP than [BHH+19] by using Ligero/Ligero++, since the two protocols reduce the proof size of ZKBoo/ZKB++, paying an overhead in the prover's work. Though [BBB+18] also proposed a proof system that achieves succinct proof size, the prover's work is still expensive (the verifier's work could be reduced via techniques in [Lee21]). As noted in [Tha22] Section 19.3.2, for circuits with small size, $O(|C| \log(|C|))$ field operations are likely to be faster than $O(|C|)$ group operations. Thus, our instantiation is likely to have better prover performance when it comes to circuits with small size. Other works in Table 1 are either private-coin or require a trusted setup to generate the common reference strings.

Table 1: Comparisons among ZKPs for composite statements

Protocols	Transparent	Prover time	Verifier time	Proof size	Form of non-algebraic part
[CGM16] Constr.2 *	✓	$O(\lambda)$ pub $O(\|C\|+\|w\|\lambda)$ sym	$O(\lambda)$ pub $O(\|C\|+\|w\|\lambda)$ sym	$O(\|w\|)\ \mathbb{G}$ $O(\|C\|)\ \mathbb{F}$	BC
[BBB+18]	✓	$O(\|C\|)$ pub	$O(\frac{\|C\|}{\log(\|C\|)})$ pub	$O(\log(\|C\|))\ \mathbb{G}$	AC
[AGM18]	✗	$O(\|C\|+\lambda)$ pub	$O(\|w\|+\lambda)$ pub	$O(1)\ \mathbb{G}$	AC
[BHH+19] Constr.2	✓	$O((\|w\|+\lambda)$ pub $O(\|C\|)$ sym	$O((\|w\|+\lambda)$ pub $O(\|C\|)$ sym	$O(\|C\|)\ \mathbb{F}$ $O(\|w\|)\ \mathbb{G}$	BC
[CFQ19] LegoUAC	✗	$O(\|C\|)$ pub	$O(\|w\|+\log^2(\|C\|))$ pub	$O(\log^2(\|C\|))\ \mathbb{G}$	AC
[ABC+22]	✗	$O(\|C\|+\|w\|)$ pub	$O(\|w\|)$ pub	$O(\log(\|w\|))\ \mathbb{G}$	AC
This work	✓	$O(\lambda)$ pub $O(\|C\|\log(\|C\|))$ sym	$O(\frac{(\|w\|+\lambda)^2}{\log(\|w\|+\lambda)})$ pub $O(\|C\|)$ sym	$O(\lambda)\ \mathbb{G}$ $O(\mathsf{polylog}(\|C\|))\ \mathbb{F}$	AC

* Means being private-coin. We use pub to indicate a public-key operation (i.e., an exponentiation or a multiplication in a cryptographic group), sym to a symmetric-key operation (i.e., an operation over a field or a hash computation). We denote by $|C|$ the circuit size, by $|w|$ the witness length, by λ the security parameter, by $\mathbb{G}$ a group element and by $\mathbb{F}$ a field element. BC is short for boolean circuit and AC is short for arithmetic circuit.

1.2 Related Work

Sigma protocols. The notion was first proposed by Cramer [Cra96] as an abstraction of Schnorr protocol [Sch91] for proving knowledge of discrete logarithm and Guillou-Quisquater protocol [GQ88] for proving knowledge of modular root. Since its introduction, Sigma protocols have received much attention due to their simplicity and high efficiency, and a great deal of works have focused on improving the efficiency of the Sigma protocols or combinations of Sigma protocols. For example, Cramer, Damgård and Schoenmakers [CDS94] applied the secret sharing technique to construct proofs of partial knowledge, i.e., given n statements $x_1, \dots, x_n$, convincing the verifier that the prover knows a witness w for at least one of the statements. Roughly speaking, in their construction, the prover shares a given challenge e into n challenges $e_1, \dots, e_n$, then uses e_i as the challenge in an individual run of the Sigma protocol for x_i. Abe et al. [AAB+20] then improved this technique by letting the prover hash the shares before using them as challenges, resulting in several significant benefits. Abe et al. [AAB+21] also introduced a model of monotone computation called *acyclicity program* (ACP), and proposed an alternative method for proving partial knowledge based on the ACP. Beullens [Beu20] introduced a new notion called *sigma protocols with helper*, referring to the Sigma protocols where the prover and the verifier are assisted by a trusted third party, and further improved the efficiency of several Sigma protocols using the new notion.

However, few works study the common design principal of Sigma protocols. To our knowledge, [Mau15] is the only work on this direction. In [Mau15], Maurer proposed a template for building Sigma protocols for algebraic statements that can be captured by preimage of a group homomorphism. Despite a large number of classic Sigma protocols can be explained by this template, it still has deficiencies in generality and utility. First, Maurer's template [Mau15] is tied to group homomorphism, and is less flexible cause it imposes fixed formats on three move messages. For instance, it fails to encompass the variant Schnorr and the batching Schnorr protocol introduced in [GLSY04] where the initial message is not computed using the same homomorphism as the statement. Second, Maurer's template does not establish connection between Sigma protocols and other cryptographic primitives. The shed light on the machinery of Sigma protocols is still unclear.

ZKPs for composite statements. This line of research started with the work of Chase, Ganesh and Mohassel [CGM16]. They gave two efficient ZKPs for proving composite statement, of which the number of expensive public-key operations is independent of the size of the circuit C. However, both of the two constructions are based on the general-purpose ZKPs from garbled circuits proposed by [JKO13], which makes the protocols interactive inherently. Agrawal, Ganesh, and Mohassel [AGM18] further presented non-interactive protocols, which use the QAP-based succinct non-interactive arguments of knowledge (SNARK) to prove the non-algebraic part of the statement. Their protocols take advantage of having a small proof size and fast verification time, while require a trusted setup for generating the structured common reference string (CRS). Bünz et al. [BBB+18] provided a protocol that can be used to prove statements expressed as arithmetic circuits and Pedersen commitments. However, the number of public key operations that the prover needs to perform is linear to the circuit size. Backes et al. [BHH+19] presented non-interactive protocols which have the merit of requiring no trusted setup, and having an efficient prover and verifier. However, their protocol makes use of the ZKBoo [GMO16]/ZKB++ [CDG+17] protocols which follow the MPC-in-the-head paradigm to prove the non-algebraic statement, thus resulting in a large proof size that is linear to $|C|$. Campanelli et al. [CFQ19] proposed a framework of ZKPs for composite statements utilizing pairing-based general-purpose ZKPs, achieving succinct proof size while requiring a trusted setup. Recently, Aranha et al. [ABC+22] proposed a general method of compiling Algebraic Holographic Proofs into ZKPs for composite statements, whose proof size is logarithmic to the number of commitments in the statements while also requiring a trusted setup.

2 Preliminaries

Notations. For an integer n, we use $[n]$ to denote the set $\{1, \ldots, n\}$. For a set X and integer t, we use $|X|$ to denote the size of X, use X_t to indicate the set consisting of all t-sized subsets of X, and use $x \xleftarrow{R} X$ to denote sampling x uniformly at random from X. We use the abbreviation PPT to indicate probabilistic polynomial-time. We denote a negligible function in λ by $\mathsf{negl}(\lambda)$. For a field $\mathbb{F}$, an integer m and a vector $\mathbf{a} = (a_i)_{i\in[m]} \in \mathbb{F}^m$, we denote by $\mathbf{V}(\mathbf{a})$ the $m \times m$ Vandermonde matrix $(v_{i,j})_{i,j\in[m]}$ where $v_{i,j} = a_i^{j-1}$ for all $i, j \in [m]$ and denote by $\mathbf{V}(\mathbf{a})^{-1}$ the inverse of this Vandermonde matrix.

2.1 Commitment Schemes

We first recall the definition of commitment schemes.

Definition 1 (Commitment Schemes). *A commitment scheme is a triple of polynomial time algorithms as below:*

- $\mathsf{Setup}(1^\lambda)$*: on input a security parameter λ, outputs the public commitment key pp, which includes the descriptions of the message space M, randomness space R, and commitment space C.*
- $\mathsf{Com}(m; r)$*: on input a message $m \in M$ and a randomness $r \in R$, outputs a commitment c.*
- $\mathsf{Verify}(c, m, r)$*: on input a commitment $c \in C$, a message $m \in M$ and a randomness $r \in R$, outputs "1" if $\mathsf{Com}(m; r) = c$ and "0" otherwise.*

Additionally, we require the following properties of a commitment scheme.

Correctness. *For any pp $\leftarrow \mathsf{Setup}(1^\lambda)$, any $m \in M$ and any $r \in R$, it holds that $\mathsf{Verify}(\mathsf{Com}(m; r), m, r) = 1$.*
Hiding. *A commitment $\mathsf{Com}(m; r)$ should reveal no information about m. Formally, it is computationally (resp. statistically) hiding if for any PPT (resp. unbounded) adversary $\mathcal{A}$, it holds that:*

$$\Pr\left[b' = b : \begin{array}{l} pp \leftarrow \mathsf{Setup}(1^\lambda); \\ (m_0, m_1) \leftarrow \mathcal{A}(pp); \\ b \xleftarrow{R} \{0,1\}, r \xleftarrow{R} R, c \leftarrow \mathsf{Com}(m_b; r); \\ b' \leftarrow \mathcal{A}(c); \end{array} \right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

Binding. *A commitment can not be opened in two different messages. Formally, it is computationally (resp. statistically) binding if for any PPT (resp. unbounded) adversary $\mathcal{A}$, it holds that:*

$$\Pr\left[\begin{array}{c} m_0 \neq m_1 \wedge \\ \mathsf{Com}(m_0; r_0) = \mathsf{Com}(m_1; r_1) \end{array} : \begin{array}{l} pp \leftarrow \mathsf{Setup}(1^\lambda); \\ (m_0, r_0, m_1, r_1) \leftarrow \mathcal{A}(pp); \end{array} \right] \leq \mathsf{negl}(\lambda).$$

Remark 1 (One-way hiding). For commitment schemes whose Com algorithms are deterministic, namely the randomness is null, we consider a weaker security notion called *one-way hiding*, which can be defined similarly as above. Roughly speaking, we say a commitment scheme is one-way hiding if the adversary only takes a negligible probability to open a randomly chosen commitment.

2.2 Sigma Protocols

Let L be an $\mathcal{NP}$ language and R be the associated binary relation. We say an instance x lies in L if and only if there exists a witness w such that $(x, w) \in R$. Consider following three-move interaction between two PPT algorithms P and V: (1) Commit: P sends an initial message to V; (2) Challenge: V sends random challenge e to P; (3) Response: P replies with a response z. A formal definition of Sigma protocols is presented as below.

Definition 2 (Sigma Protocols). *A Sigma protocol for a relation R is a three-move public-coin protocol with above communication pattern and satisfies the following three properties:*

Completeness. *If $P(x, w)$, $V(x)$ behave honestly and $(x, w) \in R$, then V always accepts the transcript.*
*n-**Special soundness.*** *There exists a PPT extraction algorithm* Ext *that on input any instance x and any n accepting transcripts $(a, e_1, z_1), \dots, (a, e_n, z_n)$ for x where all e_i's are distinct, outputs a witness w for x.*
Special honest-verifier zero-knowledge (SHVZK). *There exists a PPT simulator* Sim *that on input any instance x and any challenge e, generates a transcript (a, e, z) such that the tuple is distributed identically to an accepting transcript generated by a real protocol run between the honest $P(x, w)$ and $V(x)$.*

Lemma 1 ([ACK21, GMO16]). *Let n be a positive integer bounded by a polynomial and $\langle P, V \rangle$ be a Sigma protocol with n-special soundness. If the verifier V samples the challenge uniformly at random from the challenge space C, then $\langle P, V \rangle$ is knowledge sound with knowledge error bounded by $(n-1)/|C|$.*

2.3 Secure Multiparty Computation

A multiparty computation (MPC) protocol allows n parties $P_1, \dots, P_n$ to jointly compute an n-party function f over their inputs while maintaining the privacy of their inputs. For a set of parties $I \subseteq [n]$, we denote by f_I the outputs of parties in I after the joint computation of f. Let $view_i$ be the view of P_i during the execution of an MPC protocol, including its private input, randomness and the received messages. Below, we recall some important definitions and lemmas of MPC protocols from [IKOS07] .

Definition 3 (Consistent Views). *We say a pair of views $(view_i, view_j)$ are consistent, with respect to the protocol Π and some public input x, if the outgoing messages implicit in $view_i$, x are identical to the incoming messages reported in $view_j$ and vice versa.*

Lemma 2 (Local vs. global consistency). *Let Π be an n-party protocol, x be a public input and $view_1, \dots, view_n$ be n (possible incorrect) views. Then all pairs of views are consistent with respect to Π and x if and only if there exists an honest execution of Π with public input x (and some choice of private inputs and random inputs).*

In the semi-honest model, the security of an MPC protocol can be divided into the following two requirements.

Definition 4 (Correctness). *An MPC protocol Π realizes an n-party functionality $f(x, w_1, \dots, w_n)$ with perfect correctness, if for all inputs $x, w_1, \dots, w_n$, the probability that the outputs of some players are different from the output of f is 0.*

Definition 5 (t-privacy). *Let $1 \leq t < n$. We say an MPC protocol Π realizes an n-party functionality f with perfect t-privacy, if there exists a PPT simulator* Sim *such that for any inputs $x, w_1, \dots, w_n$, and any set of parties $I \subset [n]$ where $|I| \leq t$, the joint view of parties in I is distributed identically to* $\mathsf{Sim}(I, x, (w_i)_{i \in I}, f_I(x, w_1, \dots, w_n))$.

2.4 (Verifiable) Secret Sharing

A secret sharing (SS) scheme [Sha79] among a dealer and n participants $P_1, \ldots, P_n$ consists of two phases, called *Sharing* and *Reconstruction*. In the *Sharing* phase, the dealer shares a secret s (either a single value or a vector) among n participants, in such a way that no unauthorized subsets of participants can learn anything about the secret, while any authorized subsets of participants can recover the secret in the *Reconstruction* phase. The formal definition is as below.

Definition 6 (Secret Sharing). *A secret sharing scheme consists of three polynomial-time algorithms as follows:*

– $\mathsf{Setup}(1^\lambda)$*: on input a security parameter λ, outputs the system parameters pp, including descriptions of secret space M, share space S, the number of participants n, the privacy threshold t_p and the fault-tolerance threshold t_f, where all the three parameters n, t_p and t_f are positive integers and hold that $n \geq t_f > t_p$.*
– $\mathsf{Share}(s)$*: on input the secret $s \in M$, outputs n shares $(s_i)_{i\in[n]} \in S^n$.*
– $\mathsf{Recover}(I, (s_i)_{i\in I})$*: on input a set of participants $I \subseteq [n]$ and a vector of shares $(s_i)_{i\in I}$ where $s_i \in S$, outputs a secret $s \in M$ or a special reject symbol $\perp$ denoting failure.*

An SS scheme should satisfy the following two properties:

t_f***-Correctness.*** *In Reconstruction phase, any group of at least t_f participants can recover the secret. Formally, for any pp $\leftarrow \mathsf{Setup}(1^\lambda)$ where pp include the fault-tolerance threshold t_f, any secret $s \in M$, any $(s_i)_{i\in[n]} \leftarrow \mathsf{Share}(s)$ and any subset $I \subseteq [n]$ where $|I| \geq t_f$, it holds that $\mathsf{Recover}(I, (s_i)_{i\in I}) = s$.*
t_p***-Privacy.*** *In Sharing phase, the joint view of at most t_p participants reveals nothing about the secret. Formally, for any pp $\leftarrow \mathsf{Setup}(1^\lambda)$ where pp include the privacy threshold t_p, any $s \in M$ and any set $I \subset [n]$ where $|I| \leq t_p$, there exists a simulator Sim such that the distributions of the outputs of $\mathsf{Sim}(I)$ and $(s_i)_{i\in I}$ that generated by a real execution of $\mathsf{Share}(s)$ are identical.*

Verifiable secret sharing. Note that a secret sharing scheme is secure when the dealer and participants are all honest, while in many applications, a scheme which is able to prevent malicious behaviours from them is needed. Thereby, Chor et al. [CGMA85] put forward a stronger notion called verifiable secret sharing (VSS) schemes, where each participant is able to check the validity of the received share, such that the behavior of delivering invalid shares will be detected. Feldman [Fel87] further introduced the concept of non-interactive VSS schemes, where each participant could check the validity of his own share without interaction between other participants.

3 A Framework of Sigma Protocols From VSS

3.1 A Refined Definition of VSS Schemes

Before describing the framework, we first give a refined definition of VSS, adapted from the definition in [Fel87].

Definition 7 (Verifiable Secret Sharing). *A verifiable secret sharing scheme consists of following four algorithms:*

– $\mathsf{Setup}(1^\lambda)$*: on input the security parameter λ, outputs system parameters pp, including descriptions of secret space M, share space S, randomness space R (if there is any), commitment space C, the number of participants n, the privacy threshold t_p and the fault-tolerance threshold t_f, where all the three parameters n, t_p and t_f are positive integers and hold that $n \geq t_f > t_p$.*

– $\mathsf{Share}(s)$*: on input a secret* $s \in M$*, outputs a commitment* $c \in C$*,* n *shares* $(v_i)_{i\in[n]} \in S^n$ *and the authentication information* aut*. For ease of exposition, we describe the process by two algorithms:*

$$c \leftarrow \mathsf{Com}(s; r),$$
$$((v_i)_{i\in[n]}, aut) \leftarrow \mathsf{Share}^*(s, r),$$

where the randomness r *could be null in some settings.*
– $\mathsf{Check}(i, v_i, c, aut)$*: on input* P_i*'s index* i *and share* v_i*, a commitment* c *and the authentication information* aut*, outputs "1" iff* v_i *is valid for* P_i *w.r.t.* c *and* aut*; outputs "0", otherwise.*
– $\mathsf{Recover}(I, (v_i)_{i\in I})$*: on input a set of participants* $I \subseteq [n]$ *and a vector of shares* $(v_i)_{i\in I}$ *where* $v_i \in S$*, outputs a secret* $s \in M$ *and a randomness* $r \in R$ *(if there is any), or a special reject symbol* $\perp$ *denoting failure.*

A VSS scheme should satisfy following three properties:

Acceptance. *If the dealer honestly shares the secret, then all honest participants who receive correct shares will output "accept" in the end of Sharing phase. Formally, for any* $pp \leftarrow \mathsf{Setup}(1^\lambda)$*,* $s \in M$*,* $(c, (v_i)_{i\in[n]}, aut) \leftarrow \mathsf{Share}(s)$*, it holds that* $\mathsf{Check}(i, v_i, c, aut) = 1$ *for all* $1 \le i \le n$*.*
t_f***-Correctness.*** *Any group with at least* t_f *honest participants who output "accept" at the end of Sharing phase can recover a secret via algorithm* $\mathsf{Recover}$ *and the reconstructed secret must be an opening of the public commitment. Formally, for any* $pp \leftarrow \mathsf{Setup}(1^\lambda)$ *where* pp *include the fault-tolerance threshold* t_f*, any* $c \in C$*, any* aut *and any vector of shares* $(v_i)_{i\in I} \in S^{|I|}$ *where* $I \subseteq [n]$ *and* $|I| \ge t_f$*, if for all* $1 \le j \le m$*, it holds that* $\mathsf{Check}(i, v_i, c, aut) = 1$*, then for* $(s, r) \leftarrow \mathsf{Recover}(I, (v_i)_{i\in I})$*, it satisfies* $\mathsf{Com}(s; r) = c$*.*
t_p***-Privacy.*** *The joint view of* t_p *or less participants reveals nothing about the secret except a commitment to it. Formally, for any* $pp \leftarrow \mathsf{Setup}(1^\lambda)$ *where* pp *include the privacy threshold* t_p*, any* $s \in M$*, any* $c \leftarrow \mathsf{Com}(s; r)$ *and any set* $I \subset [n]$ *where* $|I| \le t_p$*, there exists a simulator* Sim *such that the distributions of the output of* $\mathsf{Sim}(c, I)$ *and* $((v_i)_{i\in I}, aut)$ *that generated by the real execution of* $\mathsf{Share}^*(s, r)$ *are identical.*

For notation convenience, we denote (n, t_p, t_f)-(V)SS by (verifiable) secret sharing schemes with number of participants n, privacy threshold t_p and fault-tolerance threshold t_f. Particularly, we say a verifiable secret sharing scheme VSS is with respect to a commitment scheme Com, if VSS.Share runs Com.Com as a subroutine to commit to the secret.

An alternative syntax of VSS. Our current syntax of VSS follows the commonly used definitions in the literature, i.e., making Share algorithm output shares $(v_1, ..., v_n)$ for notation convenience and clarity. An alternative syntax of VSS could be given by decomposing the algorithm Share^* into two PPT sub-algorithms as below:

(i) $\mathsf{Share\text{-}in\text{-}Mind}(s, r)$: on input a secret s and a randomness r, outputs a compact description of the sharing method SH_{cpt} and the associated authentication information aut. Both of their sizes are no larger than $\mathsf{poly}(\lambda)$.
(ii) $\mathsf{Distribute}(s, r, SH_{\mathrm{cpt}}, i)$: on input the secret s, the randomness r, the compact description of the sharing method SH_{cpt} and an index i, generates share v_i for participant P_i as per the prefixed sharing method. This step is analogous to the private key extraction algorithm in identity-based cryptography, which generates the private keys for users on-the-fly upon request.

This syntax is more general than the one in Definition 7, since all the VSS schemes meeting Definition 7 can be modified to meet this one (simply through setting $SH_{\mathrm{cpt}} = (v_1, \ldots, v_n)$), while not vice versa. In the case when n would be superpolynomial in λ[2] (e.g., the VSS schemes in Sections 4.2 to 4.4), in order to ensure the efficiency of the VSS scheme, it must follow this syntax.

[2] The value n denotes the maximum number of possible participants, it could be as large as the size of the field, which could even be exponential in security parameter λ.

Flexible design of VSS. The VSS schemes can be designed in a flexible manner. For example, when the secret s is a vector, the commitment c could either be a single vector commitment committing to the multiple entries of the secret at once (e.g., using Pedersen vector commitment [Ped91, BBB+18]), or a vector of commitments committing to each entry of the secret (e.g., the VSS scheme in Section 4.2). Meanwhile, the shares v_i's could either be packed shares of the multiple entries of s (e.g., being generated by using packed Shamir's secret sharing scheme [FY92] as the VSS scheme in Section 4.2), or be a collection of separate shares of each entry of the secret. Moreover, the authentication information aut could be viewed as a commitment to the sharing procedure, which possibly are in the form of polynomial commitments, non-interactive zero-knowledge proofs or something else.

Comparing with the definition in [Fel87]. Our refined definition has several differences from the definition in [Fel87]. In terms of syntax, there are four differences:

1. In our definition, the secret s is committed via an algorithm Com rather than being encrypted as in Feldman's definition. This change makes our definition more general, as it allows for utilizing a broader range of cryptographic techniques.
2. Our definition incorporates the committing as an integral process of sharing, while committing and sharing are treated as separate processes in Feldman's definition. This modification emphasizes the inherent connection between the committed value and the shared secret, rendering the definition more in line with the functionality of VSS.
3. The algorithm Share in our definition will additionally output authentication information aut (and a commitment c generated by its subroutine Com), while in Feldman's definition, the algorithm Share only outputs the shares. The information aut is crucial for participants to check the validity of their own shares.
4. The algorithm $\mathsf{Recover}$ in our definition will output the opening of a commitment, i.e., the secret s and the randomness r (if there is any), while in Feldman's definition, the algorithm $\mathsf{Recover}$ only outputs the secret s. Looking ahead, this modification is crucial for the security proof of our Sigma protocols framework.

In terms of security properties, there are two differences:

1. For the correctness, our definition does not stipulate that the secrets recovered by different groups of participants are consistent as in Feldman's definition, instead we require that the recovered secrets and randomness (if there is any) must be an opening of the commitment c. Looking ahead, this requirement is crucial for our application and in fact has been met by many existing VSS schemes, such as the Feldman's [Fel87] and Pedersen's VSS schemes [Ped91], but it has never been formally defined.
2. For privacy, we provide a simulation-based definition rather than a game-based one as in Feldman's definition. Such adoption makes our definition more convenient to use in the context of ZKP and MPC. In particular, the simulator Sim is given the commitment c as an auxiliary input, making the definition more general, as it allows the use of commitment schemes satisfying merely one-way hiding property.

3.2 The Framework of Sigma Protocols

Having settled a satisfactory definition of VSS, we are ready to describe our framework of Sigma protocols. Let Com $= (\mathsf{Setup}, \mathsf{Com}, \mathsf{Verify})$ be an algebraic commitment scheme, and VSS $= (\mathsf{Setup}, \mathsf{Share}, \mathsf{Check}, \mathsf{Recover})$ be an (n, t_p, t_f)-VSS scheme w.r.t Com. The framework of Sigma protocols for relation $R_{\text{Com}} = \{(x; s, r) : \mathsf{Com}(s; r) = x\}$ proceeds as below (see Figure 1 for a pictorial view).

- **Commit:** the prover P acts as the dealer running $((v_i)_{i\in[n]}, aut) \leftarrow \text{VSS}.\mathsf{Share}^*(s, r)$ "in his head", and then sends the authentication information aut to the verifier V;
- **Challenge:** V chooses a random set of participants $I \subset [n]$ subject to $|I| = t_p$, and queries P for corresponding shares;
- **Response:** P replies with the shares $(v_i)_{i\in I}$.

$$\mathsf{Com}(s;r) = x$$

$\underline{P(x;s,r)}$		$\underline{V(x)}$
$((v_i)_{i\in[n]}, aut) \leftarrow \text{VSS}.\mathsf{Share}^*(s,r)$	$\xrightarrow{\quad aut \quad}$	
	$\xleftarrow{\quad I \quad}$	$I \xleftarrow{\text{R}} [n]_{t_p}$
	$\xrightarrow{\quad (v_i)_{i\in I} \quad}$	accept iff $\forall i \in I$, $\text{VSS}.\mathsf{Check}(i, v_i, x, aut) = 1$

Fig. 1: A framework of Sigma protocols for algebraic commitments

Finally, V verifies whether $(v_i)_{i\in I}$ are valid shares for $(P_i)_{i\in I}$ w.r.t. *aut* and x, and outputs *accept* iff $\mathsf{Check}(i, v_i, x, aut) = 1$ for all $i \in I$.

Theorem 1. *Suppose* VSS *is an* (n, t_p, t_f)*-VSS scheme where* $t_f \log t_f = O(\log \lambda)$*, then the protocol described in Figure 1 is a Sigma protocol for* $\mathcal{NP}$ *relation* R_{Com} *with* $\left(\binom{t_f-1}{t_p}+1\right)$*-special soundness.*

Proof. We separately argue its completeness, special soundness and SHVZK.

Completeness. This follows readily from the acceptance property of the underlying VSS schemes.
Special Soundness. We argue this by constructing a PPT extractor Ext as below. For notation convenience, let $k = \binom{t_f-1}{t_p}+1$. Since $t_f \log t_f = O(\log \lambda)$, k is bounded by $\mathsf{poly}(\lambda)$. Given any k accepting transcripts $(aut, I_j, (v_i)_{i\in I_j})_{j\in[k]}$, where $|I_j| = t_p$ and $I_j \neq I_{j'}$ for all $j \neq j'$, first note that, there exist at least t_f distinct indices $i_1, \dots, i_{t_f} \in [n]$ along with corresponding shares $v_{i_1}, \dots, v_{i_{t_f}}$ (which are possibly not unique) subject to $\text{VSS}.\mathsf{Check}(i_j, v_{i_j}, x, aut) = 1$ for all $j \in [t_f]$. This is because if not, then there must be a (t_f-1)-sized set T, such that all I_j's are subsets of T. Since the total number of t_p-sized subsets of T is $\binom{t_f-1}{t_p} < \binom{t_f-1}{t_p}+1$, there must exist two sets $I_j = I_{j'}$ where $j \neq j'$ by the pigeonhole principle. This contradicts to the hypothesis that $I_j \neq I_{j'}$ for all $j \neq j'$. Thus, Ext can extract a witness simply through running $\text{VSS}.\mathsf{Recover}$ on input $(i_j)_{j\in[t_f]}$, $(v_{i_j})_{j\in[t_f]}$ and taking the output (s,r) as its own output. By the correctness of VSS scheme, the reconstructed witness (s,r) must hold that $\mathsf{Verify}(x,s,r) = 1$. This implies that the soundness error of the Sigma protocol in 1 is $\binom{t_f-1}{t_p}/\binom{n}{t_p}$, which is no greater than $(t_f/n)^{t_p}$.
SHVZK. We prove the SHVZK property by constructing a simulator Sim as below. Given the statement x and a challenge $I \in [n]_{t_p}$, the simulator Sim invokes the simulator of VSS scheme $\mathsf{Sim}_{\text{VSS}}$ on input (x, I) and outputs the same as $\mathsf{Sim}_{\text{VSS}}$ does, which includes the joint views of parties in I, namely the shares $(v_i)_{i\in I}$ and the authentication information *aut*. Based on the t_p-privacy of VSS scheme, the simulated transcript is distributed identically to real one.

An alternative framework. In the light of the alternative syntax of VSS in Section 3.1, there is also an alternative framework of Sigma protocols from VSS. Specifically, in the *Commit* phase, P runs $(SH_{\text{cpt}}, aut) \leftarrow \text{VSS}.\mathsf{Share\text{-}in\text{-}Mind}(s,r)$ and sends *aut* to V. In the *Challenge* phase, V chooses and sends random set $I \subset [n]$ as before. In the *Response* phase, P runs $v_i \leftarrow \text{VSS}.\mathsf{Distribute}(s, r, SH_{\text{cpt}}, i)$ for all $i \in I$. In fact, this framework could yield more efficient Sigma protocols, since the prover only needs to compute the requested shares, not all the shares. Sigma protocols in Sections 4.2 to 4.4 all follow this framework.

Parameters selection. The three parameters n, t_p, t_f of the underlying VSS schemes could be any positive integers subject to $|\mathbb{F}| \geq n \geq t_f > t_p$ where $\mathbb{F}$ is the field to which parameters n, t_p, t_f belong. However, there are two caveats that warrant attention:

1. When n is superpolynomial in the security parameter λ, the Sigma protocols from such VSS schemes follow the alternative framework as aforementioned. This is because, the underlying VSS schemes in this case must follow the alternative syntax for efficiency reasons.
2. If the soundness error $(t_f/n)^{t_p}$ in a single execution of the protocol is not negligible in the security parameter λ, one should repeat the protocol in parallel to amplify soundness. To achieve soundness error of $2^{-\lambda}$, one should set the repetition number $\rho = \frac{\lambda}{t_p(\log n - \log t_f)}$.

Size of I. For the sake of simplicity, we set the size of I to t_p, which is equal to the privacy threshold of the VSS scheme. Actually, it is possible to set the size of I to be an arbitrary positive number k smaller than t_p, thus leading to Sigma protocols with $\left(\binom{t_f-1}{k}+1\right)$-special soundness. This can be proved similarly as in the proof of Theorem 1.

4 Instantiations of Our Framework

In this section, we demonstrate the generality of our framework by recovering the classic Schnorr [Sch91], Batching Schnorr [GLSY04], Okamoto [Oka92] and GQ [GQ88] protocols from corresponding VSS schemes.

4.1 Proof of Knowledge of A Discrete Logarithm

Let $\mathbb{G}$ be a cyclic group with generator g and prime order p, define $\mathsf{Com}(s) = g^s$. Given a commitment $x \in \mathbb{G}$, we show how to prove knowledge of s such that $g^s = x$. In Section 1.1.1, we have showed how to recover the classic Schnorr protocol from Feldman's VSS scheme. Below, we present another Sigma protocol from the following additive VSS scheme.

– $\mathsf{Setup}(1^\lambda)$: runs $(\mathbb{G}, p, g) \leftarrow \mathsf{GroupGen}(1^\lambda)$, sets the total number of participants $n \le p$, the privacy threshold $t_p = n-1$ and the fault-tolerance threshold $t_f = n$, outputs $pp = ((\mathbb{G}, p, g), n, t_p, t_f)$.
– $\mathsf{Share}(s)$: computes commitment $c = g^s$, picks $s_1, \ldots, s_n \xleftarrow{\mathrm{R}} \mathbb{Z}_p$ subject to $s = \sum_{i=1}^{n} s_i \mod p$, then sets P_i's share $v_i = s_i$ and $aut = (c_1, \ldots, c_{n-1})$ where $c_i = g^{s_i}$ for $i \in [n-1]$, outputs the vector $(c, (v_i)_{i\in[n]}, aut)$.
– $\mathsf{Check}(i, v_i, c, aut)$: parses $aut = (c_1, \ldots, c_{n-1})$, if $i \in [1, n-1]$, then outputs "1" iff $g^{v_i} = c_i$ and outputs "0" otherwise; if $i = n$, then outputs "1" if $g^{v_i} = c/\prod_{j=1}^{n-1} c_j$ and "0" otherwise.
– $\mathsf{Recover}(I, (v_i)_{i\in I})$: outputs $s = \sum_{i\in I} v_i \mod p$.

Theorem 2. *The above VSS scheme satisfies acceptance, n-correctness and $(n-1)$-privacy properties.*

By plugging the above VSS scheme into our framework, we obtain a variant of Schnorr protocol for proving knowledge of a discrete logarithm (as depicted in Figure 2).

4.2 Proof of Knowledge of Several Discrete Logarithms

Define $\mathsf{Com}(\mathbf{s}) = (g^{s_j})_{j\in\{1,\ldots,|\mathbf{s}|\}}$. Given a vector of commitments $\mathbf{x} = (x_j)_{j\in[\ell]}$, we show how to prove knowledge of $\mathbf{s} = (s_j)_{j\in[\ell]}$ such that $g^{s_j} = x_j$ for all $j \in [\ell]$. Consider following VSS scheme:

– $\mathsf{Setup}(1^\lambda)$: runs $(\mathbb{G}, p, g) \leftarrow \mathsf{GroupGen}(1^\lambda)$, picks a positive number $\ell \in \mathbb{Z}_p^*$, sets the total number of participants $n \le p$ and the privacy threshold t_p and the fault-tolerance threshold $t_f = t_p + \ell$, outputs $pp = ((\mathbb{G}, p, g), n, t_p, t_f, \ell)$.
– $\mathsf{Share}(\mathbf{s})$: on input the secret $\mathbf{s} = (s_j)_{j\in[\ell]}$, runs following three algorithms and outputs $(\mathbf{c}, SH_{\text{cpt}}, aut)$:
  - $\mathsf{Com}(\mathbf{s})$: computes $c_j = g^{s_j}$ for $j \in [\ell]$, outputs $\mathbf{c} = (c_j)_{j\in[\ell]} \in \mathbb{G}^\ell$;
  - $\mathsf{Share\text{-}in\text{-}Mind}(\mathbf{s})$: selects $a_1, \ldots, a_{t_p} \xleftarrow{\mathrm{R}} \mathbb{Z}_p^*$, defines a polynomial $A(x) = \sum_{j=1}^{t_p+\ell} a_j \cdot x^{j-1}$ where $a_{t_p+j} = s_j$ for all $j \in [\ell]$, sets $SH_{\text{cpt}} = (a_j)_{j\in[t_p]}$ and $aut = (\widetilde{c}_j)_{j\in[t_p]}$ where $\widetilde{c}_j = g^{a_j}$ for $j \in [t_p]$, outputs (SH_{cpt}, aut);

$$
\begin{array}{ccc}
 & x = g^s & \\
\underline{P(x;s)} & & \underline{V(x)} \\
\begin{array}{c} s_1,\ldots,s_n \xleftarrow{\mathrm{R}} \mathbb{Z}_p \\ \text{s.t. } s = \sum_{i=1}^{n} s_i \mod p \\ \text{for } i \in [n-1],\ c_i = g^{s_i} \end{array} & \xrightarrow{\quad aut = (c_1,\ldots,c_{n-1}) \quad} & \\
 & \xleftarrow{\qquad I \qquad} & I \xleftarrow{\mathrm{R}} [n]_{n-1} \\
 & \xrightarrow{\qquad (s_i)_{i\in I} \qquad} & \begin{array}{c} \text{accept iff for } i \in I, \\ \text{if } i \in [1,n-1],\ g^{s_i} = c_i, \\ \text{if } i = n,\ g^{s_i} = x/\prod_{j=1}^{n-1} c_j \end{array}
\end{array}
$$

Fig. 2: A Sigma protocol for proving knowledge of a discrete logarithm

- Distribute($\mathbf{s}, SH_{\text{cpt}}, i$): parses $\mathbf{s} = (s_j)_{j\in[\ell]}$ and $SH_{\text{cpt}} = (a_j)_{j\in[t_p]}$, sets $a_{t_p+j} = s_j$ for $j \in [\ell]$, computes $v_i = \sum_{j=1}^{t_p+\ell} a_j \cdot i^{j-1} \mod p$, outputs v_i. (This algorithm is run upon request.)

– Check($i, v_i, \mathbf{c}, aut$): parses $\mathbf{c} = (c_j)_{j\in[\ell]}$ and $aut = (\widetilde{c}_j)_{j\in[t_p]}$, outputs "1" if it holds that $g^{v_i} = \left(\prod_{j=1}^{t_p} \widetilde{c}_j^{\,i^{j-1}}\right) \cdot \left(\prod_{j=1}^{\ell} c_j^{\,i^{t_p+j-1}}\right)$ and "0" otherwise.

– Rec($I, (v_i)_{i\in I}$): computes a polynomial $A(x)$ such that $A(i) = v_i$ for all $i \in I$, sets s_j be the $(t_p + j)$-th coefficient of A, outputs $(s_j)_{j\in[\ell]}$.

Theorem 3. *Above VSS scheme satisfies acceptance, $(t_p + \ell)$-correctness and t_p-privacy.*

By plugging the above VSS scheme into our framework, we obtain a Sigma protocol for proving knowledge of several discrete logarithms (as depicted in Figure 3). By setting parameters $n = p$ and $t_p = 1$, we recover the Batching Schnorr protocol [GLSY04].

$$
\begin{array}{ccc}
 & x_1 = g^{s_1},\ldots,x_\ell = g^{s_\ell} & \\
\underline{P((x_j)_{j\in[\ell]};(s_j)_{j\in[\ell]})} & & \underline{V((x_j)_{j\in[\ell]})} \\
\begin{array}{c} a_1,\ldots,a_{t_p} \xleftarrow{\mathrm{R}} \mathbb{Z}_p^* \\ \text{for } j \in [t_p],\ \widetilde{c}_i = g^{a_i} \end{array} & \xrightarrow{\quad aut = (\widetilde{c}_j)_{j\in[t_p]} \quad} & \\
 & \xleftarrow{\qquad I \qquad} & I \xleftarrow{\mathrm{R}} [n]_{t_p} \\
\begin{array}{c} \text{for } i \in I, \\ v_i = \sum_{j=1}^{t_p} a_j \cdot i^{j-1} + \sum_{j=1}^{\ell} s_j \cdot i^{t_p+j-1} \mod p \end{array} & \xrightarrow{\qquad (v_i)_{i\in I} \qquad} & \begin{array}{c} \text{accept iff for } i \in I, \\ g^{v_i} = \prod_{j=1}^{t_p} \widetilde{c}_j^{\,i^{j-1}} \cdot \prod_{j=1}^{\ell} x_j^{\,i^{t_p+j-1}} \end{array}
\end{array}
$$

Fig. 3: A Sigma protocol for proving knowledge of several discrete logarithms

4.3 Proof of Knowledge of A Representation

Define $\mathsf{Com}(s;r) = g^s h^r$ where g, h are two different generators of group $\mathbb{G}$. Given a commitment x, we show how to prove knowledge of (s,r) such that $g^s h^r = x$ from the Pedersen's VSS scheme [Ped91] as below:

- $\mathsf{Setup}(1^\lambda)$: runs $(\mathbb{G}, p, g, h) \leftarrow \mathsf{GroupGen}(1^\lambda)$, sets the total number of participants $n \leq p$, the privacy threshold t_p and the fault-tolerance threshold $t_f = t_p + 1$, outputs $pp = ((\mathbb{G}, p, g, h), n, t_p, t_f)$.
- $\mathsf{Share}(s)$: on input the secret s, runs following three algorithms and outputs $(c, SH_{\text{cpt}}, aut)$:
  - $\mathsf{Com}(s;r)$: picks a random element $r \xleftarrow{\text{R}} \mathbb{Z}_p^*$, outputs $c = g^s h^r$;
  - $\mathsf{Share\text{-}in\text{-}Mind}(s,r)$: picks two random t_p-degree polynomials $A(x) = \sum_{i=0}^{t_p} a_i \cdot x^i$ and $B(x) = \sum_{i=0}^{t_p} b_i \cdot x^i$ subject to $a_{t_p} = s$ and $b_{t_p} = r$, computes $c_j = g^{a_j} h^{b_j}$ for $0 \leq j \leq t_p - 1$, sets $SH_{\text{cpt}} = (a_j, b_j)_{0 \leq j \leq t_p - 1}$ and $aut = (c_j)_{0 \leq j \leq t_p - 1}$, outputs (SH_{cpt}, aut);
  - $\mathsf{Distribute}(s, r, SH_{\text{cpt}}, i)$: parses $SH_{\text{cpt}} = (a_j, b_j)_{0 \leq j \leq t_p - 1}$, sets $a_{t_p} = s$ and $b_{t_p} = r$, computes $s_i = \sum_{j=0}^{t_p} a_j \cdot i^j \mod p$ and $r_i = \sum_{j=0}^{t_p} b_j \cdot i^j \mod p$, outputs $v_i = (s_i, r_i)$. (This algorithm is run upon request.)
- $\mathsf{Check}(i, v_i, c, aut)$: parses $v_i = (s_i, r_i)$ and $aut = (c_j)_{0 \leq j \leq t_p - 1}$, outputs "1" if $g^{s_i} h^{r_i} = c^{i^{t_p}} \cdot \prod_{j=0}^{t_p - 1} c_j^{i^j}$ and "0" otherwise.
- $\mathsf{Recover}(I, (v_i)_{i \in I})$: parses $v_i = (s_i, r_i)$, constructs two polynomials $A(x)$, $B(x)$ such that $A(i) = s_i$ and $B(i) = r_i$ for all $i \in I$, sets s be the coefficient of the t_p-degree term of A and r be that of B, outputs (s, r).

Theorem 4. *Pedersen's VSS scheme satisfies acceptance, (t_p+1)-correctness and t_p-privacy.*

By plugging the above VSS scheme into our framework, we obtain a Sigma protocol for proving knowledge of a representation (as depicted in Figure 4). By setting parameters $n = p$ and $t_p = 1$, we recover the classic Okamoto protocol [Oka92].

$$x = g^s h^r$$

$\underline{P(x;s,r)}$		$\underline{V(x)}$
$a_0, \ldots, a_{t_p-1} \xleftarrow{\text{R}} \mathbb{Z}_p^*$; $b_0, \ldots, b_{t_p-1} \xleftarrow{\text{R}} \mathbb{Z}_p^*$; for $0 \leq j \leq t_p - 1$, $c_j = g^{a_j} h^{b_j}$	$\xrightarrow{aut = (c_j)_{0 \leq j \leq t_p - 1}}$	
	$\xleftarrow{I}$	$I \xleftarrow{\text{R}} [n]_{t_p}$
for $i \in I$, compute $s_i = s \cdot i^{t_p} + \sum_{j=0}^{t_p-1} a_j \cdot i^j \mod p$ $r_i = r \cdot i^{t_p} + \sum_{j=0}^{t_p-1} b_j \cdot i^j \mod p$	$\xrightarrow{(s_i, r_i)_{i \in I}}$	accept iff $\forall i \in I$, $g^{s_i} h^{r_i} = x^{i^{t_p}} \cdot \prod_{j=0}^{t_p-1} c_j^{i^j}$

Fig. 4: A Sigma protocol for proving knowledge of a representation

4.4 Proof of Knowledge of An eth Root

Let GenRSA be a PPT algorithm that on input security parameter λ, outputs an RSA public key (N, e), where e is prime. Given an $x \in \mathbb{Z}_N^*$, we show how to prove knowledge of s such that $x = s^e \mod N$ from following VSS scheme:

– $\mathsf{Setup}(1^\lambda)$: runs $(N; e) \leftarrow \mathsf{GenRSA}(1^\lambda)$, where e is prime, sets the total number of participants $n \leq e$, and sets the privacy threshold $t_p = 1$ and the fault-tolerance threshold $t_f = 2$, outputs $pp = ((N, e), n, t_p, t_f)$.
– $\mathsf{Share}(s)$: on input a secret $s \in \mathbb{Z}_N^*$, runs following three algorithms and outputs $(c, SH_{\text{cpt}}, aut)$:
  - $\mathsf{Com}(s)$: computes the commitment $c = s^e \mod N$, outputs c;
  - $\mathsf{Share\text{-}in\text{-}Mind}(s)$: picks a random element $a \in \mathbb{Z}_N^*$, defines a function $f(x) = a \cdot s^x \mod N$, sets $SH_{\text{cpt}} = a$, computes $aut = a^e \mod N$, outputs (SH_{cpt}, aut);
  - $\mathsf{Distribute}(s, SH_{\text{cpt}}, i)$: parses $SH_{\text{cpt}} = a$, computes $s_i = a \cdot s^i \mod N$, outputs $v_i = s_i$. (This algorithm is run upon request.)
– $\mathsf{Check}(i, v_i, c, aut)$: outputs "1" if $v_i^e = aut \cdot c^i \mod N$ and "0" otherwise.
– $\mathsf{Recover}(I, (v_i)_{i \in I}, c)$: if $|I| < 2$ outputs $\perp$; else, runs the extended Euclidean algorithm yields integers α, $\beta \in \mathbb{Z}_N^*$ such that $\alpha \cdot e + \beta \cdot (i_2 - i_1) = 1$, outputs $s = c^\alpha (v_{i_2}/v_{i_1})^\beta \mod N$.

Theorem 5. *The above VSS scheme satisfies acceptance, 2-correctness and 1-privacy properties.*

By plugging the above VSS scheme into our framework, we obtain a Sigma protocol for proving knowledge of an e-th root (as depicted in Figure 5). By setting the parameter $n = e$, we recover the classic GQ protocol [GQ88].

$$x = s^e \mod N$$

$$\begin{array}{ccc}
\underline{P(x; s)} & & \underline{V(x)} \\
a \xleftarrow{\mathrm{R}} \mathbb{Z}_N^* & \xrightarrow{\quad aut \quad} & \\
aut = a^e \mod N & & i \xleftarrow{\mathrm{R}} [n] \\
 & \xleftarrow{\quad i \quad} & \\
v_i = a \cdot s^i \mod N & \xrightarrow{\quad v_i \quad} & \text{accept iff} \\
 & & v_i^e = aut \cdot x^i \mod N
\end{array}$$

Fig. 5: A Sigma protocol for proving knowledge of an e-th root

5 A Framework of ZKPs for Composite Statements

In this section, we are going to show the application of the Sigma protocols from VSS in giving a generic construction of efficient ZKPs for composite statements. In this work, we focus on a common form of composite statement where given a commitment x and a value y, the prover wants to prove the knowledge of (s, r) such that $\mathsf{Com}(s; r) = x \wedge C(s) = y$, where C is an arithmetic/boolean circuit. In a nutshell, we use Sigma protocols from VSS to prove the algebraic parts, use ZK protocols from MPC to prove the non-algebraic parts, and enforce consistency between the witnesses used in two parts via *witness sharing reusing*.

5.1 A Generalization of MPC-in-the-Head Paradigm

Before designing the framework of ZKPs for composite statements, we first generalize the MPC-in-the-head paradigm introduced by Ishai et al. [IKOS07] via extending the XOR-based secret sharing scheme to an (n, t_p, t_f)-SS scheme. Precisely, to construct a ZK protocol for $\mathcal{NP}$ relation $R_C = \{(y; s) : C(s) = y\}$ using

MPC-in-the-head technique, we need three building blocks: a secret sharing scheme, an MPC protocol and a commitment scheme.

Let SS = ($\mathsf{Setup}, \mathsf{Share}, \mathsf{Recover}$) be an $(n, t_p^{\mathrm{ss}}, t_f)$-secret sharing scheme, $\widehat{\mathrm{Com}} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{Verify})$ be a commitment scheme and Π_f be a t_p^{mpc}-private n-party protocol that realizes a n-party function f, where $f(y, s_1, \dots, s_n) = 1$ if and only if $C(\mathsf{Recover}([n], (s_i)_{i\in[n]})) = y$, and integers $t_p^{\mathrm{mpc}} < t_p^{\mathrm{ss}}$. Then, ZK protocols following MPC-in-the-head paradigm proceeds as below (as depicted in Figure 6):

- **Commit:** the prover P shares the witness s into n shares $s_1, \dots, s_n$ by running SS.$\mathsf{Share}(s)$, then runs MPC protocol Π_f "in his head" with shares $s_1, \dots, s_n$ as input of n virtual parties, then commits to each party's share s_i and view $view_i$ (without loss of generality, we separate P_i's input s_i from his view $view_i$ and concatenate them with notation $||$), and sends the n commitments to V;
- **Challenge:** V picks a random t_p^{mpc}-sized subset I of $[n]$ and sends it to P;
- **Response:** P opens corresponding commitments through revealing corresponding shares and views to V.

Finally, V outputs "accept" iff the three conditions listed hereunder hold:

1. the commitments are successfully opened;
2. all the outputs of participants in I are "1", which are determined by their inputs s_i and views $view_i$;
3. all the opened views are consistent with each other with respect to y and Π_f.

$$
\begin{array}{ccc}
 & C(s) = y & \\
\underline{P(y;s)} & & \underline{V(y)} \\
\begin{array}{c} (s_i)_{i\in[n]} \leftarrow \mathrm{SS}.\mathsf{Share}(s) \\ (s_i||view_i)_{i\in[n]} \leftarrow \Pi_{f(y,s_1,\dots,s_n)} \\ \forall i \in [n], c_i \leftarrow \widehat{\mathrm{Com}}.\mathsf{Com}(s_i||view_i) \end{array} & \begin{array}{c} \xrightarrow{\quad c_1 \dots, c_n \quad} \\ \\ \xleftarrow{\quad I \quad} \end{array} & I \xleftarrow{\mathrm{R}} [n]_{t_p^{\mathrm{mpc}}} \\
 & \xrightarrow{\quad (s_i||view_i)_{i\in I} \quad} & \begin{array}{c} \text{accept iff } \forall i \in I \\ \widehat{\mathrm{Com}}.\mathsf{Verify}(c_i, s_i||view_i) = 1 \\ \Pi_{f_y}(P_i, s_i||view_i) = 1 \\ s_i||view_i \text{ is consistent with others} \end{array}
\end{array}
$$

Fig. 6: ZK protocols from MPC-in-the-head paradigm

Theorem 6. *Let $n > 2$, $t_p^{\mathrm{ss}} \geq t_p^{\mathrm{mpc}}$, $t_p^{\mathrm{mpc}} \cdot \log n = O(\log \lambda)$, and R_C, f be as above. Suppose SS is an $(n, t_p^{\mathrm{ss}}, t_f)$-secret sharing scheme, the MPC protocol Π_f realizes the n-party functionality f with correctness and t_p^{mpc}-privacy and $\widehat{\mathrm{Com}}$ is a commitment scheme, then the protocol in Figure 6, is a Sigma protocol for relation R_C with $\left(\binom{n-2}{t_p^{\mathrm{mpc}}} + 2\binom{n-2}{t_p^{\mathrm{mpc}}-1} + 1\right)$-special soundness.*

Proof. We separately argue its completeness, special soundness and SHVZK.

Completeness. If $(y, s) \in R_C$ and the prover is honest, then by the correctness of $\widehat{\mathrm{Com}}$, the requested commitments are always opened successfully. By the correctness of the secret sharing scheme SS, the shares

$s_1, \ldots, s_n$ hold that $\mathsf{Recover}([n], (s_i)_{i\in[n]}) = s$ and thus $f(y, s_1, \ldots, s_n) = 1$. Then, by the correctness of Π_f, the views $s_1||view_1, \ldots, s_n||view_n$ always have output "1". Besides, since the views are produced by an honest execution of the Π_f, by Lemma 2, they are all consistent with each other.
Special Soundness. For notation convenience, let $k = \binom{n-2}{t_{\mathrm{mpc}}} + 2\binom{n-2}{t_{\mathrm{mpc}}-1} + 1$. We argue the k-special soundness by constructing a PPT extractor Ext that can extract a witness s such that $(y, s) \in R_C$, given any k accepting transcripts with the same initial message and different challenges. Since $t_p^{\mathrm{mpc}} \cdot \log n = O(\log \lambda)$, k is bounded by $\mathsf{poly}(\lambda)$. In following argument, we assume for simplicity that both binding of commitment scheme and correctness of MPC protocol never fail.

Consider k accepting transcripts $((c_i)_{i\in[n]}, I_j, (s_i||view_i)_{i\in I_j})_{j\in[k]}$: first note that since the k sets $I_1, \ldots, I_k$ are distinct t_p^{mpc}-sized subset of $[n]$ and $k > \binom{n-1}{t_p^{\mathrm{mpc}}}$, we have $\cup_{j=1}^k I_j = \{1, \ldots, n\}$, which implies that for all $1 \le i \le n$, P_i's view $s_i||view_i$ is revealed at least once. (This is because if not, namely if there exists at least one index $i \in [n]$ such that $i \notin \cup_{j=1}^k I_j$, then all I_j's are t_p^{mpc}-sized subsets of the set $[n] \setminus i$. Since the total number of such subsets is only $\binom{n-1}{t_p^{\mathrm{mpc}}} < k$, there must exist two sets $I_j = I_{j'}$ where $j \neq j'$ by pigeonhole principle. This contradicts to the hypothesis that the k sets $(I_j)_{j\in[k]}$ are distinct.) Thanks to the binding property of $\widehat{\mathrm{Com}}$, for the same index i, all $s_i||view_i$'s revealed in different transcripts are identical. Thereby, the k transcripts provide unique n views $s_1||view_1, \ldots, s_n||view_n$ and the extractor Ext is able to compute a witness s through running the efficient algorithm $\mathsf{Recover}([n], (s_i)_{i\in[n]}) = s$. Below, we argue that s is a valid witness such that $C(s) = y$.

First note that for each accepting transcript $((c_i)_{i\in[n]}, I, (s_i||view_i)_{i\in I})$, it indicates that for each $i \in I$, the output of player P_i with respect to Π_f, and $s_i||view_i$ is "1", and $s_i||view_i$, $s_j||view_j$ are consistent for all $i, j \in I$. Then, we prove that in the n-tuple views $s_1||view_1, \ldots, s_n||view_n$ provided by the k accepting transcripts, all pairs of views $s_i||view_i$, $s_j||view_j$ are consistent with each other. Namely, for all $i, j \in [n]$, there is a set $I \in \{I_1, \ldots, I_k\}$ such that $i \in I \wedge j \in I$. The reason is that, if there is a pair of $i', j' \in [n]$ which does not hold above, then for each set $I \in \{I_1, \ldots, I_k\}$, it must be one of the following cases:

- $i' \notin I \wedge j' \notin I$: there are $\binom{n-2}{t_p^{\mathrm{mpc}}}$ cases;
- $i' \in I \wedge j' \notin I$: there are $\binom{n-2}{t_p^{\mathrm{mpc}}-1}$ cases;
- $i' \notin I \wedge j' \in I$: there are $\binom{n-2}{t_p^{\mathrm{mpc}}-1}$ cases.

However, the total number of the above cases is only $\binom{n-2}{t_p^{\mathrm{mpc}}} + 2\binom{n-2}{t_p^{\mathrm{mpc}}-1}$, which is smaller than k. By the pigeonhole principle, there must be two identical sets in $I_1, \ldots, I_k$, which contradicts the hypothesis that all of them are distinct. Therefore, for all pairs $i, j \in [n]$, there must be a set $I \in \{I_1, \ldots, I_k\}$ such that $i \in I \wedge j \in I$ and thus all pairs of views $s_i||view_i$, $s_j||view_j$ are consistent with each other. By Lemma 2, the n-tuple views provided by the k accepting transcripts correspond to an honest execution of Π_f. Moreover, by the correctness of Π_f, in an honest execution of Π_f on input $s_1, \ldots, s_n$, the probability that the output of some player is different from the output of f is 0. Besides, as mentioned before, all the revealed views $s_i||view_i$'s are such that the output of P_i is "1". Thus, it holds that $f(y, s_1, \ldots, s_n) = 1$ and as the definition of f, we have $C(\mathsf{Recover}([n], (s_j)_{j\in[n]}) = C(s) = y$.
SHVZK. Let $\mathsf{Sim}_{\mathrm{SS}}$ and Sim_{Π_f} be the simulators for the underlying secret sharing scheme SS and MPC protocol Π_f, respectively. We prove SHVZK by constructing a PPT simulator $\mathsf{Sim}(y, I)$ where $I \subset [n]$ and $|I| = t_p^{\mathrm{mpc}}$, by invoking $\mathsf{Sim}_{\mathrm{SS}}$ and Sim_{Π_f} as below:

1. Run $\mathsf{Sim}_{\mathrm{SS}}$ on input I, receiving a vector of shares $(s_i)_{i\in I}$.
2. Run Sim_{Π_f} on input $(I, y, (s_i)_{i\in I}, 1)$, receiving t_p^{mpc} views $(s_i||view_i)_{i\in I}$.
3. For $i \in [n] \wedge i \notin I$, select random string $str_i \xleftarrow{\mathrm{R}} \{0,1\}^{|s||view|}$ and set $s_i||view_i = str_i$.
4. For $i \in [n]$, compute $\widehat{\mathrm{Com}}.\mathsf{Com}(s_i||view_i) \to c_i$.
5. Output $((c_i)_{i\in[n]}, I, (s_i||view_i)_{i\in I})$.

Then we show that the transcripts output by Sim are indistinguishable from transcripts of real executions of the protocol with an honest verifier via a sequence of hybrid transcripts as follows:

– Hybrid_0: Real transcript.
– Hybrid_1: Same as Hybrid_0, except that the simulator is given a random challenge $I \xleftarrow{R} [n]_{t_p^{\text{mpc}}}$ in advance, and for $i \in [n] \wedge i \notin I$, it selects random string $str_i \xleftarrow{R} \{0,1\}^{|w||view|}$, and computes $c_i \leftarrow \widehat{\mathsf{Com}}.\mathsf{Com}(str_i)$.
– Hybrid_2: Same as Hybrid_1, except that the simulator runs Sim_{Π_f} on input $(I, y, (s_i)_{i\in I}, 1)$, obtaining the simulated views $(s_i||view_i)_{i\in I}$.
– Hybrid_3: Same as Hybrid_2, except that the simulator is not provided the witness and instead it runs Sim_{SS} on input I and obtains a vector of shares $(s_i)_{i\in I}$.

Since the commitments $(c_i)_{i\in[n]\wedge i\notin I}$ are never opened, the indistinguishability of Hybrid_0 and Hybrid_1 follows directly from the hiding property of the commitment scheme. The indistinguishability of Hybrid_1 and Hybrid_2 follows from the t_p^{mpc}-privacy property of Π_f. Since $t_p^{\text{mpc}} \leq t_p^{\text{ss}}$, The indistinguishability of Hybrid_2 and Hybrid_3 follows straightforwardly from the t_p^{ss}-privacy property of the underlying secret sharing scheme SS.

5.2 Separable VSS Schemes

As discussed in Section 1, in order to combine Sigma protocols from VSS and ZK protocols from MPC seamlessly, we are interested in VSS schemes which satisfy a mild property called *Separability*. Since the parameter n in the MPC-in-the-head paradigm is bounded by $\mathsf{poly}(\lambda)$, we consider the separability of VSS schemes simply using the syntax in Definition 7. Informally, for a VSS scheme, we say it satisfies *Separability* if the following two conditions hold:

1. The algorithm $\mathsf{Share}^*(s, r)$ could be separated into two sub-algorithms, one for generating the shares $(v_i)_{i\in[n]}$ and the other for generating the authentication information aut. Particularly, the shares $(v_i)_{i\in[n]}$ are generated as per some secret sharing schemes and aut is generated by committing to the sharing method (i.e., the shares $(v_i)_{i\in[n]}$ in the syntax in Definition 7 or the compact description of the sharing method SH_{cpt} in the alternative syntax).
2. If the randomness r is not a dummy value, then each share v_i could be divided into two values s_i and r_i, where the former is a share of the secret s and the later is a share of the randomness r. That is, s and r are secret-shared separately.

Below, we formally define the *Separability* property.

Definition 8 (Separability). *For an (n, t_p, t_f)-VSS scheme* VSS*, we say it satisfies separability if there is an (n, t_p, t_f)-SS scheme* SS *and an algorithm* AutGen *such that the algorithms* $\text{VSS}.\mathsf{Share}^*$ *and* $\text{VSS}.\mathsf{Recover}$ *can be separated as below:*

$$\begin{aligned}
\text{VSS}.\mathsf{Share}^*(s,r): &\ (s_i)_{i\in[n]} \leftarrow \text{SS}.\mathsf{Share}(s)\\
&\ (r_i)_{i\in[n]} \leftarrow \text{SS}.\mathsf{Share}(r)\\
&\ aut \leftarrow \mathsf{AutGen}((s_i, r_i)_{i\in[n]})\\
&\ \mathit{return}\ ((s_i, r_i)_{i\in[n]}, aut)
\end{aligned}$$

$$\begin{aligned}
\text{VSS}.\mathsf{Recover}(I, (v_i)_{i\in I}): &\ \forall 1 \leq j \leq |I|,\ \text{parse } v_i = (s_i, r_i)\\
&\ s \leftarrow \text{SS}.\mathsf{Recover}(I, (s_i)_{i\in[n]})\\
&\ r \leftarrow \text{SS}.\mathsf{Recover}(I, (r_i)_{i\in[n]})\\
&\ \mathit{return}\ (s, r)
\end{aligned}$$

If r is null, then only the s will be secret-shared and recovered.

Particularly, we say a VSS scheme is *with respect to* an SS scheme if it generates the shares as per this SS scheme.

Remark 2. More generally, in such separable VSS schemes, the SS schemes used to share secret s and randomness r could be different in some settings.

5.3 Generic Construction of ZKPs for Composite Statements

Now, we proceed to describe the generic construction of ZKPs for composite statements. Formally, let Com be an algebraic commitment algorithm and C be an arbitrary circuit, we give a zero-knowledge proof for relation:

$$R_{\text{cs}} = \{(x, y; s, r) : \mathsf{Com}(s; r) = x \wedge C(s) = y\}.$$

Let Π_C^{MPC} be a Sigma protocol for $\{(y; s) : C(s) = y\}$ from MPC as depicted in Figure 6 and using building blocks: an $(n, t_p^{\text{ss}}, t_f)$-SS scheme SS, a commitment scheme $\widehat{\text{Com}}$, and a t_p^{mpc}-private n-party protocol Π_f. Let $\Pi_{\text{Com}}^{\text{VSS}}$ be a Sigma protocol for $\{(x; s, r) : \mathsf{Com}(s; r) = x\}$ following the framework as in Figure 1 and using building blocks: an $(n, t_p^{\text{vss}}, t_f)$-VSS scheme VSS w.r.t. Com and SS. Below, we show how to obtain a ZK protocol $\Pi_{\text{Com},C}$ for composite statements through combining Π_C^{MPC} and $\Pi_{\text{Com}}^{\text{VSS}}$, which is also a Sigma protocol. The full protocol is presented in Figure 7 and the overlap between Π_C^{MPC} and $\Pi_{\text{Com}}^{\text{VSS}}$ are highlighted in rectangles.

- **Commit:** P proceeds as in $\Pi_{\text{Com}}^{\text{VSS}}$, running algorithm $((s_i, r_i)_{i\in[n]}, aut) \leftarrow \text{VSS}.\mathsf{Share}^*(s, r)$, which can be separated into three algorithms $(s_i)_{i\in[n]} \leftarrow \text{SS}.\mathsf{Share}(s)$, $(r_i)_{i\in[n]} \leftarrow \text{SS}.\mathsf{Share}(r)$ and $aut \leftarrow \mathsf{AutGen}((s_i, r_i)_{i\in[n]})$. Then, P proceeds as in Π_C^{MPC} while reusing the shares $(s_i)_{i\in[n]}$. Next, P sends $c_1, \ldots, c_n$ and aut to V.
- **Challenge:** V picks a random t_{mpc}-sized subset I of $[n]$ as in Π_C^{MPC}.
- **Response:** P responds with participants' inputs and views $(s_i || view_i)_{i\in I}$ as in Π_C^{MPC} and shares of randomness $(r_i)_{i\in I}$ as in $\Pi_{\text{Com}}^{\text{VSS}}$.

Finally, V outputs "accept" iff $(s_i || view_i)_{i\in I}$ pass the verification of Π_C^{MPC} and $(s_i, r_i)_{i\in I}$ pass the verification of $\Pi_{\text{Com}}^{\text{VSS}}$.

$$\mathsf{Com}(s; r) = x \wedge C(s) = y$$

$$\begin{array}{ccc}
\underline{P(x, y; s, r)} & & \underline{V(x, y)} \\
\boxed{(s_i)_{i\in[n]} \leftarrow \text{SS}.\mathsf{Share}(s)} & & \\
(r_i)_{i\in[n]} \leftarrow \text{SS}.\mathsf{Share}(r) & & \\
aut \leftarrow \mathsf{AutGen}((s_i, r_i)_{i\in[n]}) & \xrightarrow{c_1 \ldots, c_n, aut} & \\
(s_i || view_i)_{i\in[n]} \leftarrow \Pi_{f(y, s_1, \ldots, s_n)} & & \\
\forall i \in [n], c_i \leftarrow \widehat{\text{Com}}.\mathsf{Com}(s_i || view_i) & & I \xleftarrow{\text{R}} [n]_{t_p^{\text{mpc}}} \\
& \xleftarrow{I} & \\
& \xrightarrow{(r_i, \boxed{s_i} || view_i)_{i\in I}} & \text{accept iff } \forall i \in I, \\
& & \widehat{\text{Com}}.\mathsf{Verify}(c_i, s_i || view_i) = 1 \\
& & \Pi_f(P_i, s_i || view_i) = 1 \\
& & s_i || view_i \text{ is consistent with others} \\
& & \text{VSS}.\mathsf{Check}(i, (s_i, r_i), x, aut) = 1
\end{array}$$

Fig. 7: A ZK protocol for composite statements

Theorem 7. *Let $n > 2$, $t_p^{\mathrm{ss}} \geq t_p^{\mathrm{mpc}}$, $t_p^{\mathrm{mpc}} \cdot \log n = O(\log \lambda)$. Suppose the protocol Π_C^{MPC} constructed as in Figure 6 using building blocks* SS, $\widehat{\mathrm{Com}}$ *and Π_f as above, is a Sigma protocol for relation $\{(y; s) : C(s) = y\}$ with $\left(\binom{n-2}{t_p^{\mathrm{mpc}}} + 2\binom{n-2}{t_p^{\mathrm{mpc}}-1} + 1\right)$-special soundness, protocol $\Pi_{\mathrm{Com}}^{\mathrm{VSS}}$ constructed as in Figure 1 using building block* VSS *which is with respect to* Com *and* SS, *is a Sigma protocol for relation $\{(x; s, r) : \mathsf{Com}(s; r) = x\}$, then the protocol $\Pi_{\mathrm{Com},C}$ constructed as in Figure 7 is a Sigma protocol for R_{cs} with $\left(\binom{n-2}{t_p^{\mathrm{mpc}}} + 2\binom{n-2}{t_p^{\mathrm{mpc}}-1} + 1\right)$-special soundness.*

Proof. We separately argue its completeness, special soundness and SHVZK.

Completeness. We first argue the correctness of protocol $\Pi_{\mathrm{Com},C}$. For an honestly generated transcript $((c_1, \ldots, c_n, aut), I, (r_i, s_i||view_i)_{i \in I})$, it is obvious that $((c_1, \ldots, c_n), I, (s_i||view_i)_{i \in I})$ is actually an honestly generated transcript of Π_C^{MPC}, thus by the correctness of Π_C^{MPC}, the verifier will pass the first three checks. And $(aut, I, (s_i, r_i)_{i \in I})$ is an honestly generated transcript of $\Pi_{\mathrm{Com}}^{\mathrm{VSS}}$, thus by the correctness of $\Pi_{\mathrm{Com}}^{\mathrm{VSS}}$ the verifier will pass the last check, and finally outputs "accept".
Special soundness. For notation convenience, let $k = \binom{n-2}{t_{\mathrm{mpc}}} + 2\binom{n-2}{t_{\mathrm{mpc}}-1} + 1$. We argue the k-special soundness by constructing a PPT extractor Ext that can extract a valid witness $w = (s, r)$, given any k accepting transcripts with the same initial message and different challenges. Since $t_p^{\mathrm{mpc}} \cdot \log n = O(\log \lambda)$, k is bounded by $\mathsf{poly}(\lambda)$. Likewise, we assume for simplicity that both binding of $\widehat{\mathrm{Com}}$ and correctness of Π_f never fail.

First note that, both Π_C^{MPC} and $\Pi_{\mathrm{Com}}^{\mathrm{VSS}}$ satisfy k-special soundness. The former has been proved in Theorem 6. As shown in the proof of Theorem 6, the PPT extractor Ext_C works as follows: on input k accepting transcripts as required, runs SS.$\mathsf{Recover}([n], (s_i)_{i \in [n]}) = s$ where $(s_i)_{i \in [n]}$ is uniquely determined by the k accepting transcripts, outputs a witness s such that $C(s) = y$. The latter can be proved similarly as the proof of Theorem 1. Since VSS is with respect to SS, there exists a PPT extractor $\mathsf{Ext}_{\mathrm{Com}}$ proceeds as follows: on input k accepting transcripts as required, runs SS.$\mathsf{Recover}([n], (s_i)_{i \in [n]}) = s$ and SS.$\mathsf{Recover}([n], (r_i)_{i \in [n]}) = r$ where the shares $(s_i, r_i)_{i \in [n]}$ are subject to VSS.$\mathsf{Check}(i, (s_i, r_i), x, aut) = 1$ for all $i \in [n]$, and then outputs (s, r). By the correctness of VSS, it holds that $\mathsf{Com}(s; r) = x$.

Next, we show how the extractor Ext makes use of Ext_C and $\mathsf{Ext}_{\mathrm{Com}}$ to extract a witness $w = (s, r)$ such that $(x, y; s, r) \in R_{\mathrm{cs}}$. Concretely, on input k accepting transcripts $((c_i)_{i \in [n]}, aut), I_j, (r_i, s_i||view_i)_{i \in I_j})_{j \in [k]}$, Ext works as below:

1. Run Ext_C on input $((c_i)_{i \in [n]}, I_j, (s_i||view_i)_{i \in I_j})_{j \in [k]}$ and obtain s;
2. Run $\mathsf{Ext}_{\mathrm{Com}}$ on input $(aut, I_j, (r_i, s_i)_{i \in I_j})_{j \in [k]}$ and obtain (s', r);
3. outputs (s, r).

Since both s and s' are output by the deterministic algorithm SS.$\mathsf{Recover}$ on the same input $([n], (s_i)_{i \in [n]})$ (where $(s_i)_{i \in [n]}$ is uniquely determined by the k accepting transcripts), we have $s = s'$. Furthermore, by the k-special soundness of Π_C^{MPC} and $\Pi_{\mathrm{Com}}^{\mathrm{VSS}}$, it holds that $\mathsf{Com}(s; r) = x \wedge C(s) = y$.
SHVZK. We prove this by constructing a PPT simulator Sim that on input any tuple $((x, y), I)$ where $|I| \leq t_p^{\mathrm{mpc}}$ could simulate the transcript of $\Pi_{\mathrm{Com},C}$.

First note that, both Π_C^{MPC} and $\Pi_{\mathrm{Com}}^{\mathrm{VSS}}$ satisfy the SHVZK property. The former has been proved in Theorem 6. As shown in the proof of Theorem 6, the PPT simulator $\mathsf{Sim}_C(y, I)$ where $|I| \leq t_p^{\mathrm{mpc}}$ generates $(s_i)_{i \in I}$ essentially by running $(s_i)_{i \in I} \leftarrow \mathsf{Sim}_{\mathrm{SS}}(I)$. The latter has been proved in Theorem 1. As shown in the proof Theorem 1, the PPT simulator $\mathsf{Sim}_{\mathrm{Com}}(x, I)$ where $|I| \leq t_p^{\mathrm{vss}}$ simulates the transcripts essentially by running $((s_i, r_i)_{i \in I}, aut) \leftarrow \mathsf{Sim}_{\mathrm{VSS}}(x, I)$. Furthermore, since VSS is with respect to SS, we have $t_p^{\mathrm{vss}} = t_p^{\mathrm{ss}}$ and the simulator $\mathsf{Sim}_{\mathrm{VSS}}(x, I)$ generates $(s_i)_{i \in I}$ by running $(s_i)_{i \in I} \leftarrow \mathsf{Sim}_{\mathrm{SS}}(I)$ as well.

Next, we show how Sim makes use of Sim_C and $\mathsf{Sim}_{\mathrm{Com}}$ to simulate a transcript for $\Pi_{\mathrm{Com},C}$. Concretely, on input $((x, y), I)$ where $|I| \leq t_p^{\mathrm{mpc}} \leq t_p^{\mathrm{ss}}$, it works as below:

1. Run $\mathsf{Sim}_C(y, I)$ and obtain $((c_i)_{i \in [n]}, I, (s_i||view_i)_{i \in I})$;
2. Run $\mathsf{Sim}_{\mathrm{Com}}(x, I)$, and obtain $(aut, I, (s_i, r_i)_{i \in I})$, where $(s_i)_{i \in I}$ is a reuse of $(s_i)_{i \in I}$ that generated in Step 1, rather than a new one generated by running $\mathsf{Sim}_{\mathrm{SS}}(I)$ a second time;

3. Output $(((c_i)_{i\in[n]}, aut), I, (r_i, s_i||view_i)_{i\in I})$.

By the SHVZK property of Π_C^{MPC} and $\Pi_{\mathrm{Com}}^{\mathrm{VSS}}$, it is straightforward that the distribution of Sim's output is indistinguishable with a real transcription of $\Pi_{\mathrm{Com},C}$.

Remark 3 (Key element required for combining). In order to get better efficiency, some practical protocols in the MPC-in-the-head paradigm slightly deviate from the template in Section 5.1, depending on the concrete MPC protocols they used. For example, the KKW protocol [KKW18] utilizes an MPC protocol designed in the preprocessing model and the Ligero [AHIV17]/Ligero++ [BFH+20] protocols make use of a particular type of MPC protocols in the malicious model. Nevertheless, they all retain the secret sharing procedure (though different secret sharing schemes are employed), which is the key element that is required for combining with our Sigma protocols framework in Section 3.2.

6 An Instantiation of ZKP for Composite Statements

In this section, we give a ZK protocol for composite statements by instantiating the underlying MPC-in-the-head protocol with Ligero++ [BFH+20]. Let $\mathbb{F}_p$ be a large prime field and $C : \mathbb{F}_p^m \to \mathbb{F}_p$ be an arithmetic circuit. We show how to prove following composite statements: given a vector of Pedersen commitments $\mathbf{x} = (x_1, \ldots, x_m)$, the prover wants to convince the verifier that he knows the witness $(\mathbf{s}, \mathbf{r}) \in \mathbb{F}_p^m \times \mathbb{F}_p^m$ such that $C(\mathbf{s}) = 1 \wedge x_i = g^{s_i} h^{r_i}$ for $1 \le i \le m$.

As we have noticed, in order to construct a ZK protocol for composite statements using Ligero++, the key point is giving a VSS scheme that is with respect to the SS scheme used by Ligero++, and then constructing a Sigma protocol from it.

6.1 Review of Ligero++

We briefly recall the Ligero++ protocol and analyze the SS scheme it uses. (Notably, Ligero++ uses the same SS scheme as Ligero [AHIV17].) At a high level, to prove knowledge of $\mathbf{s} = (s_i)_{i\in[m]} \in \mathbb{F}_p^m$ such that $C(\mathbf{s}) = 1$, the Ligero++ prover first generates an extended witness which contains the circuit input $\mathbf{s}$ and the outputs of $|C|$ gates, then arranges the extended witness in a matrix of size $\frac{C}{\mathsf{polylog}|C|} \times \mathsf{polylog}|C|$ (where the first m entries are $(s_i)_{i\in[m]}$) and encodes each row using Reed-Solomon (RS) Code. The verifier challenges the prover to reveal the linear combinations of the rows of the codeword matrix, and checks its consistency through invoking inner-product argument (IPA) protocols on $\widetilde{t}$ randomly picked columns. As mentioned in [BFH+20], to remain zero-knowledge during the consistency check, it is desirable to either utilize zero-knowledge IPA protocols or make the encoding randomized. For further consideration, we use a randomized RS encoding to ensure zero knowledge. The formal definition of RS code is presented below.

Definition 9 (Reed-Solomon Code). *For positive integers n, k, a finite field $\mathbb{F}$, and a vector $\eta = (\eta_1, \ldots, \eta_n)$ of distinct elements of $\mathbb{F}$, the code $\mathsf{RS}_{\mathbb{F},n,k,\eta}$ is the $[n, k, n-k+1]$ linear code over $\mathbb{F}$ that consists of all n-tuples $(P(\eta_1), \ldots, P(\eta_n))$ where P is a polynomial of degree $< k$ over $\mathbb{F}$.*

Definition 10 (Encoded message). *Let $L = \mathsf{RS}_{\mathbb{F},n,k,\eta}$ be an RS code and $\zeta = (\zeta_1, \ldots, \zeta_\ell)$ be a vector of distinct elements of $\mathbb{F}$ for $\ell \le k$. For a codeword $u = (u_1, \ldots, u_n) \in L$, we say it encodes (or rather, can be decodes to) the message $(P_u(\zeta_1), \ldots, P_u(\zeta_\ell))$, where P_u is the polynomial (of degree $< k$) corresponding to u.*

Encoding & **Sharing.** We can simply make the RS code $\mathsf{RS}_{\mathbb{F},n,k,\eta}$ randomized via increasing the degree of polynomials by $\widetilde{t}$ where $\widetilde{t} < k$, and it is evident that the randomized RS code $\mathsf{RS}_{\mathbb{F},n,k,\eta}$ can be viewed as the (variant) packed Shamir's SS scheme [FY92] with number of participants n, privacy threshold $t_p = \widetilde{t}$ and the fault-tolerance $t_f = k$. That is, encoding a message is equivalent to sharing the message: to encode (resp., share) a message $(s_i)_{i\in[\ell]}$ using randomized $\mathsf{RS}_{\mathbb{F},n,k,\eta}$ (resp., packed Shamir's SS scheme), one first selects $\widetilde{t}$ random elements $\alpha_1, \ldots, \alpha_{\widetilde{t}} \in \mathbb{F}$ where $\ell + \widetilde{t} = k$ and generates a polynomial $P(x)$ with degree $< \ell + \widetilde{t}$ such that $P(\zeta_i) = s_i$ for all $i \in [\ell]$ and $P(\zeta_{\ell+i}) = \alpha_i$ for all $i \in [\widetilde{t}]$, then sets the codeword (resp., shares) to be $(P(\eta_1), \ldots, P(\eta_n))$. Therefore, the codeword matrix aforementioned is also the shares matrix.

Modifications to Ligero++. As mentioned before, the Ligero++ protocol does not strictly conform to the generalized MPC-in-the-head paradigm in Section 5.1, due to the different MPC model it used. There are two main differences that could pose challenges in combining Ligero++ with Sigma protocols. First, the witness to be shared is an expanded version that encompasses the input of circuit and the outputs of all circuit gates, rather than only the input itself, making the shares opened later be an expanded version as well. Second, the $\widetilde{t}$ random columns of shares matrix will not be opened directly due to the invocation of IPA protocols, causing obstructions of reusing witness shares. Fortunately, both of them can be overcame with a few modifications to Ligero++: dividing the shares matrix into two vertically concatenated sub-matrices and handling them differently when in the consistency check. Specifically, the two sub-matrices and their respective handling methods are as follows:

- The first sub-matrix is the first m/ℓ rows of the shares matrix (WLOG., we assume $m = c \cdot \ell$ for some integer $c > 0$), which in fact is the shares of circuit input $\mathbf{s}$. When in the consistency check, the prover opens its $\widetilde{t}$ entries directly to the verifier and the verifier computes the inner product of these entries with random vectors directly. Thereby, the shares of circuit input $\mathbf{s}$ could be reused later. Since the encoding is randomized, the openings leak nothing about the witness.
- The second sub-matrix is the remaining rows of the shares matrix, which are the shares of outputs of gates. When in the consistency check, the prover inputs its $\widetilde{t}$ entries on IPA protocols as originally while the inner product checked in IPA protocols should be modified according to the opened entries of the first sub-matrix.

By doing so, the shares of inputs $\mathbf{s}$ and shares of gates' outputs are separated. Moreover, it makes witness shares reusing available while maintaining the advantage of utilizing IPA technique.

6.2 A Sigma Protocol for Pedersen Commitments

Having specified the SS scheme that Ligero++ employs, we are now ready to present a VSS scheme that is with respect to this SS scheme and later give a corresponding Sigma protocol from it.

Since the parameter n in the SS scheme used by Ligero++ is bounded by $\mathsf{poly}(\lambda)$, we describe the VSS scheme simply using the syntax in Definition 7. The VSS scheme consists following four algorithms:

- $\mathsf{Setup}(1^\lambda)$: runs $(\mathbb{G}, p, g, h) \leftarrow \mathsf{GroupGen}(1^\lambda)$, sets the total number of participants n, the privacy threshold t_p, the fault-tolerance threshold threshold $t_f = \ell + t_p$, picks two disjoint vectors $\boldsymbol{\zeta} = (\zeta_j)_{j\in[\ell+t_p]} \in \mathbb{F}_p^{\ell+t_p}$ and $\boldsymbol{\eta} = (\eta_i)_{i\in[n]} \in \mathbb{F}_p^n$, and both $\boldsymbol{\zeta}, \boldsymbol{\eta}$ contain distinct elements, outputs $pp = ((\mathbb{G}, p, g, h), n, t_p, t_f, \ell, \boldsymbol{\zeta}, \boldsymbol{\eta})$.
- $\mathsf{Share}(\mathbf{s})$: on input a vector of secret $\mathbf{s} = (s_j)_{j\in[\ell]} \in \mathbb{F}_p^\ell$, runs following two algorithms and outputs $(\mathbf{c}, (v_i)_{i\in[n]}, aut)$:
  - $\mathsf{Com}(\mathbf{s}; \mathbf{r})$: selects a vector of randomness $\mathbf{r} = (r_j)_{j\in[\ell]} \xleftarrow{\text{R}} \mathbb{F}_p^\ell$, outputs a vector of commitments $\mathbf{c} = (c_j)_{j\in[\ell]}$, where $c_j = g^{s_j} h^{r_j}$ for all $j \in [\ell]$.
  - $\mathsf{Share}^*(\mathbf{s}, \mathbf{r})$: chooses two random vectors $(\alpha_j)_{j\in[t_p]}, (\beta_j)_{j\in[t_p]} \xleftarrow{\text{R}} \mathbb{F}_p^{t_p}$, interpolates two polynomials $A(x)$ and $B(x)$ such that

$$\begin{aligned} &\forall 1 \le j \le \ell,\ A(\zeta_j) = s_j,\ B(\zeta_j) = r_j; \\ &\forall \ell + 1 \le j \le \ell + t_p,\ A(\zeta_j) = \alpha_{j-\ell},\ B(\zeta_j) = \beta_{j-\ell}, \end{aligned} \tag{1}$$

    outputs shares $(v_i)_{i\in[n]}$ where $v_i = (A(\eta_i), B(\eta_i))$ for all $i \in [n]$ and $aut = (\widetilde{c}_j)_{j\in[t_p]}$ where $\widetilde{c}_j = g^{\alpha_j} h^{\beta_j}$ for all $j \in [t_p]$.
- $\mathsf{Check}(i, v_i, \mathbf{c}, aut)$: parses $v_i = (v_{i1}, v_{i2})$ and $aut = (\widetilde{c}_j)_{j\in[t_p]}$, computes $h_k = \left(\prod_{j=1}^{\ell} c_j^{\delta_{k,j}}\right) \cdot \left(\prod_{j=1}^{t_p} \widetilde{c}_j^{\delta_{k,\ell+j}}\right)$ for $k \in [\ell + t_p]$, where the matrix $(\delta_{k,j})_{1\le k,j\le \ell+t_p}$ is equal to $\mathbf{V}(\boldsymbol{\zeta})^{-1}$, outputs "1" if $g^{v_{i1}} h^{v_{i2}} = \prod_{k=1}^{\ell+t_p} h_k^{\eta_i^{k-1}}$ and "0" otherwise.
- $\mathsf{Recover}(I, (v_i)_{i\in I})$: parses $v_i = (v_{i1}, v_{i2})$, uses Lagrange Interpolation to compute polynomials $A(x)$ and $B(x)$ such that $A(\eta_i) = v_{i1}$ and $B(\eta_i) = v_{i2}$ for all $i \in I$, outputs $(\mathbf{s}, \mathbf{r})$ where $(s_j, r_j) = (A(\zeta_j), B(\zeta_j))$ for $j \in [\ell]$.

Theorem 8. *The VSS scheme described above satisfies acceptance, $(\ell + t_p)$-correctness and t_p-privacy.*

By plugging the above VSS scheme into the framework in Section 1.1.1, we obtain a Sigma protocol (as depicted in Figure 8) for proving knowledge of openings of several Pedersen commitments.

$$x_1 = g^{s_1} h^{r_1}, \ldots, x_\ell = g^{s_\ell} h^{r_\ell}$$

$$\begin{array}{ccc}
\underline{P((x_j)_{j\in[\ell]}; (s_j, r_j)_{j\in[\ell]})} & & \underline{V((x_j)_{j\in[\ell]})} \\
\begin{array}{c} (\alpha_j)_{j\in[t_p]}, (\beta_j)_{j\in[t_p]} \xleftarrow{\mathrm{R}} \mathbb{F}_p^{t_p} \\ \text{compute } A(x), B(x) \text{ as in (1)} \\ \forall\, i \in [n]\text{: } s_i = A(\eta_i),\ r_i = B(\eta_i) \\ \forall\, j \in [t_p],\ \widetilde{c}_j = g^{\alpha_j} h^{\beta_j} \end{array} & \xrightarrow{\quad aut = (\widetilde{c}_j)_{j\in[t_p]} \quad} & \\
 & \xleftarrow{\quad I \quad} & I \xleftarrow{\mathrm{R}} [n]_{t_p} \\
 & \xrightarrow{\quad (s_i, r_i)_{i\in I} \quad} & \begin{array}{c} (\delta_{k,j})_{1\le k,j\le \ell+t_p} = \mathbf{V}(\boldsymbol{\zeta})^{-1} \\ \forall k \in [\ell + t_p],\ h_k = \prod_{j=1}^{\ell} x_j^{\delta_{k,j}} \cdot \prod_{j=\ell+1}^{\ell+t_p} \widetilde{c}_{j-\ell}^{\,\delta_{k,j}} \\ \text{accept iff } \forall i \in I,\ g^{s_i} h^{r_i} = \prod_{k=1}^{\ell+t_p} h_k^{\eta_i^{k-1}} \end{array}
\end{array}$$

Fig. 8: A Sigma protocol for Pedersen commitments

Parameters selection. In order to combine with Ligero++, some of the public parameters of above VSS scheme, including p, n, t_p, ℓ, $\boldsymbol{\zeta}$ and $\boldsymbol{\eta}$, should be in line with that of Ligero++. Since Ligero++ performs interpolation and evaluation using fast Fourier transform (FFT), above VSS scheme should be implemented using elliptic curves whose scalar fields $\mathbb{F}_p$ are FFT-friendly. One can refer to [AHG22] for a suitable elliptic curve.

Security analysis. Based on Lemma 1, Theorem 1 and Theorem 8, it is straightforward that the protocol in Figure 8 is a Sigma protocol with soundness error $\binom{t_f-1}{t_p}/\binom{n}{t_p}$. When setting $n = c \cdot t_f$ for some constant $c \geq 1$, we must set $t_p = \lambda/\log c$ to achieve a soundness error of $2^{-\lambda}$ without repetition. Since $\binom{t_f-1}{t_p}/\binom{n}{t_p}$ is smaller than the soundness error of Ligero++, the soundness error of ZK protocols for composite statements, obtained by combining Sigma protocols in Figure 8 and Ligero++, is dominated by the soundness error of Ligero++.

Efficiency analysis. Let λ be the security parameter, $\ell_{\mathbb{G}}$ be the length of a group element, $\ell_{\mathbb{F}}$ be the length of a field element and $\ell = |\mathbf{x}|$ be the number of commitments in the statement. Fix parameters n, t_p, t_f where $n = c{\cdot}t_f$ for some constant $c \geq 1$ and $t_f = \ell{+}t_p$. Then, the proof size is $t_p{\cdot}\ell_{\mathbb{G}}{+}2t_p{\cdot}\ell_{\mathbb{F}})$, which asymptotically is $O(\lambda)$. The prover's work includes the computations of c_k's, which need $O(t_p)$ group operations; interpolation and evaluation of polynomials, which need $O((\ell{+}t_p){\cdot}\log(\ell{+}t_p)$ field operations by using FFT. The verifier's work includes the computations of matrix $(\delta_{k,j})$, which need $O((\ell + t_p)^2)$ field operations; the computations of h_k's, which need $O(\ell + t_p)$ multi-exponentiations of size $\ell + t_p$; and the computations in the verification equations, which need $O(t_p)$ multi-exponentiations of size $\ell + t_p$. (Pippenger's [Pip80] algorithm could be used to accelerate the computations of multi-exponentiations.)

Having given the Sigma protocol for Pedersen commitments, it is not difficult to combine it with the Ligero++ protocol and get a ZK protocol for composite statements, following the method in Section 5.3, and we omit the details in this paper. The efficiency of the final ZK protocol reported in Table 1 is obtained by directly summing the costs of Ligero++ and above Sigma protocol. Since the underlying SS components are identical in Ligero and Ligero++, the Sigma protocol could also be combined with Ligero seamlessly by choosing appropriate parameters. This will lead to a faster prover while a larger proof size.

7 Conclusion

Sigma protocols are the most efficient ZKPs for proving knowledge of openings of algebraic commitments, which are defined as relations over algebraic groups. They have now become an important building block for a variety of cryptosystems. In this work, we presented a framework of Sigma protocols for algebraic statements from verifiable secret sharing schemes. This framework neatly explains the design principle underlying those classic Sigma protocols, including the Schnorr, Batching Schnorr, GQ and Okamoto protocol. In addition, it gives a generic construction of Sigma protocols for proving knowledge of algebraic commitments, thus being able to lead to new Sigma protocols that were not previously known. Furthermore, we also showed its application in designing ZKPs for composite statements. By using the *witness sharing reusing* technique, we combined the Sigma protocols from VSS and general-purpose ZKPs following MPC-in-the-head paradigm seamlessly, yielding a generic construction of ZKPs for composite statements which enjoys the advantages of having a fast prover. Through instantiating the underlying general-purpose ZKPs with Ligero++ and tailoring a corresponding Sigma protocol, we obtain a new ZK protocol for composite statements, which achieves a new balance between running time and the proof size, thus resolving the open problem left by Backes et al. (PKC 2019).

References

[AAB+20] Masayuki Abe, Miguel Ambrona, Andrej Bogdanov, Miyako Ohkubo, and Alon Rosen. Non-interactive composition of sigma-protocols via share-then-hash. In *ASIACRYPT*, 2020.

[AAB+21] Masayuki Abe, Miguel Ambrona, Andrej Bogdanov, Miyako Ohkubo, and Alon Rosen. Acyclicity programming for sigma-protocols. In *TCC*, 2021.

[ABC+22] Diego F. Aranha, Emil Madsen Bennedsen, Matteo Campanelli, Chaya Ganesh, Claudio Orlandi, and Akira Takahashi. ECLIPSE: enhanced compiling method for pedersen-committed zksnark engines. In *PKC*, 2022.

[ACK21] Thomas Attema, Ronald Cramer, and Lisa Kohl. A compressed $\varsigma $-protocol theory for lattices. In *CRYPTO*, 2021.

[AGM18] Shashank Agrawal, Chaya Ganesh, and Payman Mohassel. Non-interactive zero-knowledge proofs for composite statements. In *CRYPTO*, 2018.

[AHG22] Diego F. Aranha, Youssef El Housni, and Aurore Guillevic. A survey of elliptic curves for proof systems. *IACR Cryptol. ePrint Arch.*, 2022.

[AHIV17] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In *ACM CCS*, 2017.

[BBB+18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Gregory Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *IEEE S&P*, 2018.

[BBHR18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. 2018. http://eprint.iacr.org/2018/046.

[BCC+16] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In *EUROCRYPT*, 2016.

[BCG+14] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *IEEE S&P*, 2014.

[BCR+19] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for R1CS. In *EUROCRYPT*, 2019.

[BCS16] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. In *TCC*, 2016.

[Beu20] Ward Beullens. Sigma protocols for mq, PKP and sis, and fishy signature schemes. In *EUROCRYPT*, 2020.

[BFH+20] Rishabh Bhadauria, Zhiyong Fang, Carmit Hazay, Muthuramakrishnan Venkitasubramaniam, Tiancheng Xie, and Yupeng Zhang. Ligero++: A new optimized sublinear iop. In *ACM CCS*, 2020.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *STOC*, 1988.

[BHH+19] Michael Backes, Lucjan Hanzlik, Amir Herzberg, Aniket Kate, and Ivan Pryvalov. Efficient non-interactive zero-knowledge proofs in cross-domains without trusted setup. In *PKC*, 2019.

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *STOC*, 1988.

[CDG+17] Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. Post-quantum zero-knowledge and signatures from symmetric-key primitives. In *ACM CCS*, 2017.

[CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *CRYPTO*, 1994.

[CF85] Josh D. Cohen and Michael J. Fischer. A robust and verifiable cryptographically secure election scheme (extended abstract). In *FOCS*, 1985.

[CFQ19] Matteo Campanelli, Dario Fiore, and Anaïs Querol. Legosnark: Modular design and composition of succinct zero-knowledge proofs. In *ACM CCS*, 2019.

[CGM16] Melissa Chase, Chaya Ganesh, and Payman Mohassel. Efficient zero-knowledge proof of algebraic and non-algebraic statements with applications to privacy preserving credentials. In *CRYPTO*, 2016.

[CGMA85] Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults. In *FOCS*, 1985.

[CHM+20] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas P. Ward. Marlin: Preprocessing zksnarks with universal and updatable SRS. In *EUROCRYPT*, 2020.

[CL01] Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *EUROCRYPT*, 2001.

[COS20] Alessandro Chiesa, Dev Ojha, and Nicholas Spooner. Fractal: Post-quantum and transparent recursive proofs from holography. In *EUROCRYPT*, 2020.

[Cra96] Ronald Cramer. Modular design of secure yet practical cryptographic protocols. *PhD thesis*, 1996.

[CZ21] Hongrui Cui and Kaiyi Zhang. A simple post-quantum non-interactive zero-knowledge proof from garbled circuits. In *Inscrypt*, 2021.

[Fel87] Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *FOCS*, 1987.

[FFS87] Uriel Feige, Amos Fiat, and Adi Shamir. Zero knowledge proofs of identity. In *STOC*, 1987.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: practical solutions to identification and signature problems. In *CRYPTO*, 1986.

[FY92] Matthew K. Franklin and Moti Yung. Communication complexity of secure computation (extended abstract). In *STOC*, 1992.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct nizks without pcps. In *EUROCRYPT*, 2013.

[GK15] Jens Groth and Markulf Kohlweiss. One-out-of-many proofs: Or how to leak a secret and spend a coin. In *EUROCRYPT*, 2015.

[GLSY04] Rosario Gennaro, Darren Leigh, Ravi Sundaram, and William S. Yerazunis. Batching schnorr identification scheme with applications to privacy-preserving authorization and low-bandwidth communication devices. In *ASIACRYPT*, 2004.

[GMO16] Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. Zkboo: Faster zero-knowledge for boolean circuits. In *USENIX*, 2016.

[GMR85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *STOC*, 1985.

[GMW86] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to prove all NP-statements in zero-knowledge, and a methodology of cryptographic protocol design. In *CRYPTO*, 1986.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC*, 1987.

[GQ88] Louis C. Guillou and Jean-Jacques Quisquater. A "paradoxical" indentity-based signature scheme resulting from zero-knowledge. In *CRYPTO*, 1988.

[Gro09] Jens Groth. Linear algebra with sub-linear zero-knowledge arguments. In *CRYPTO*, 2009.

[Gro10] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In *ASIACRYPT*, 2010.

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In *EUROCRYPT*, 2016.

[HKR19] Max Hoffmann, Michael Klooß, and Andy Rupp. Efficient zero-knowledge arguments in the discrete log setting, revisited. In *ACM CCS*, 2019.

[IKO07] Yuval Ishai, Eyal Kushilevitz, and Rafail Ostrovsky. Efficient arguments without short pcps. In *IEEE CCC*, 2007.

[IKOS07] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In *STOC*, 2007.

[JKO13] Marek Jawurek, Florian Kerschbaum, and Claudio Orlandi. Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently. In *ACM CCS*, 2013.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *STOC*, 1992.

[KKW18] Jonathan Katz, Vladimir Kolesnikov, and Xiao Wang. Improved non-interactive zero knowledge with applications to post-quantum signatures. In *ACM CCS*, 2018.

[KR08] Yael Tauman Kalai and Ran Raz. Interactive PCP. In *ICALP*, 2008.

[Lee21] Jonathan Lee. Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments. In *TCC*, 2021.

[Lip12] Helger Lipmaa. Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In *TCC*, 2012.

[Mau15] Ueli Maurer. Zero-knowledge proofs of knowledge for group homomorphisms. *DCC*, 2015.

[MBKM19] Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. Sonic: Zero-knowledge snarks from linear-size universal and updatable structured reference strings. In *ACM CCS*, 2019.

[Mic94] Silvio Micali. CS proofs (extended abstracts). In *FOCS*, 1994.

[NY90] Moni Naor and Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *STOC*, 1990.

[Oka92] Tatsuaki Okamoto. Provably secure and practical identification schemes and corresponding signature schemes. In *CRYPTO*, 1992.

[Ped91] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO*, 1991.

[Pip80] Nicholas Pippenger. On the evaluation of powers and monomials. *SIAM Journal on Computing*, 1980.

[RB89] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In *STOC*, 1989.

[RRR16] Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. Constant-round interactive proofs for delegating computation. In *STOC*, 2016.

[Sah99] Amit Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *FOCS*, 1999.

[Sch91] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 1991.

[Set20] Srinath Setty. Spartan: Efficient and general-purpose zksnarks without trusted setup. In *CRYPTO*, 2020.

[Sha79] Adi Shamir. How to share a secret. *CACM*, 1979.

[Tha22] Justin Thaler. Proofs, arguments, and zero-knowledge. *Foundations and Trends® in Privacy and Security*, 2022.

[ZXZS20] Jiaheng Zhang, Tiancheng Xie, Yupeng Zhang, and Dawn Song. Transparent polynomial delegation and its applications to zero knowledge proof. In *IEEE S&P*, 2020.

A Missing Proofs

A.1 A Proof for Theorem 2

Proof. We separately argue the three properties.

Acceptance. For all honestly generated $(v_i)_{i\in[n]}$, c and $aut = (c_1, \ldots, c_{n-1})$, it always holds that $g^{v_i} = c_i$ for all $i \in [n-1]$ and $g^{v_n} = g^{s_n} = g^{s-\sum_{i=1}^{n-1} s_i} = g^s / \prod_{j=1}^{n-1} g^{s_j} = c/\prod_{j=1}^{n-1} c_j$.
***n*-Correctness**. For any commitment c, $aut = (c_1, \ldots, c_{n-1})$, participants $I = \{1, \ldots, n\}$ and shares $(v_i)_{i\in[n]}$, if it holds that $\mathsf{Check}(i, v_i, c, aut) = 1$ for all $1 \le i \le n$ (namely $g^{v_i} = c_i$ for all $1 \le i \le n-1$ and $g^{v_i} = c/\prod_{j=1}^{n-1} c_j$ for $i = n$), then it is evident that $g^{\sum_{i=1}^{n} v_i} = \prod_{i=1}^{n} g^{v_i} = c$. Thus, $s = \sum_{i=1}^{n} v_i$ is an opening for c.
$(n-1)$-privacy. We argue this by constructing the simulator $\mathsf{Sim}(c, I)$. In the two different cases, it works as follows:

1. Case I: $n \notin I$. Without loss of generality, we assume $I = \{1, \ldots, n-1\}$. Sim simulates the shares for $(P_i)_{i\in I}$, by picking $s_1, \ldots, s_{n-1} \xleftarrow{\text{R}} \mathbb{Z}_p$, and setting $v_i = s_i$. Then compute $c_i = g^{s_i}$ and set the authentication information $aut = (c_1, \ldots, c_{n-1})$. Output $((v_i)_{i\in[n-1]}, aut)$.
2. Case II: $n \in I$. Without loss of generality, we assume $I = \{2, \ldots, n\}$. Sim simulates the shares for $(P_i)_{i\in I}$, by picking $(s_i)_{i\in[2,n]} \xleftarrow{\text{R}} \mathbb{Z}_p$, and setting $v_i = s_i$. Then compute $c_i = g^{s_i}$ for $i \in [2, n]$ and $c_1 = c/(\prod_{j=2}^{n} c_j)$, set the authentication information $aut = (c_1, \ldots, c_{n-1})$. Output $((v_i)_{i\in[2,n]}, aut)$.

It is direct that the distributions of the outputs of simulator $\mathsf{Sim}(c, I)$ are identical to that of the outputs of algorithm $\mathsf{Share}^*(s)$ for participants in I.

A.2 A Proof for Theorem 3

Proof. We separately argue the three properties.

Acceptance. For all honestly generated v_i, $\mathbf{c} = (c_j)_{j\in[\ell]}$ and $aut = (\widetilde{c}_j)_{j\in[t_p]}$, it always holds that $g^{v_i} = g^{A(i)} = g^{\sum_{j=1}^{t_p+\ell} a_j \cdot i^{j-1}} = \prod_{j=1}^{t_p+\ell} g^{a_j \cdot i^{j-1}} = \left(\prod_{j=1}^{t_p} \widetilde{c}_j^{\,i^{j-1}}\right) \cdot \left(\prod_{j=1}^{\ell} c_j^{\,i^{t_p+j-1}}\right)$.
$(t_p + \ell)$-Correctness. For any set of participants $I \subset [n]$ where $|I| = m \ge t_p + \ell$ and corresponding shares $(v_i)_{i\in I}$, if it holds that $\mathsf{Check}(i, v_i, c, aut) = 1$ for all $i \in I$, then we have $g^{v_i} = \left(\prod_{j=1}^{t_p} \widetilde{c}_j^{\,i^{j-1}}\right) \cdot \left(\prod_{j=1}^{\ell} c_j^{\,i^{t_p+j-1}}\right)$ for all $i \in I$. Let $A(x) = \sum_{k=1}^{m} a_k \cdot x^{k-1}$ be the polynomial that interpolates the points $(i, v_i)_{i\in I}$. Note that, the degree of $A(x)$ is at most $m-1$. Then, it holds that $a_{t_p+j} = \sum_{i=1}^{m} \delta_{t_p+j,i} \cdot v_i \mod p$ for $j \in [\ell]$, where the vector $(\delta_{t_p+j,i})_{i\in[m]}$ is the $(t_p + j)$-th row of the matrix $\mathbf{V}(1, \ldots, m)^{-1}$. Thus, for $j \in [\ell]$, we have

$$
\begin{aligned}
g^{a_{t_p+j}} =& g^{\sum_{i=1}^{m} \delta_{t_p+j,i} \cdot v_i} \\
=& \prod_{i=1}^{m} \left(g^{v_i}\right)^{\delta_{t_p+j,i}} \\
=& \prod_{i=1}^{m} \left(\left(\prod_{k=1}^{t_p} \widetilde{c}_k^{\,i^{k-1}} \right) \cdot \left(\prod_{k=1}^{\ell} c_k^{\,i^{t_p+k-1}} \right) \right)^{\delta_{t_p+j,i}} \\
=& \left(\prod_{k=1}^{t_p} \widetilde{c}_k^{\,\sum_{i=1}^{m} i^{k-1} \cdot \delta_{t_p+j,i}} \right) \cdot \left(\prod_{k=1}^{\ell} c_k^{\,\sum_{i=1}^{m} i^{t_p+k-1} \cdot \delta_{t_p+j,i}} \right) \\
=& c_j
\end{aligned}
$$

t_p-privacy. We argue this by constructing the simulator $\mathsf{Sim}(\mathbf{c}, I)$. Without loss of generality, we assume $I = [t_p]$. The simulator $\mathsf{Sim}(\mathbf{c}, I)$ proceeds as follows:

1. Simulate the shares for $P_1, \ldots, P_{t_p}$, by picking $v_1, \ldots, v_{t_p} \xleftarrow{R} \mathbb{Z}_p$;
2. Compute the authentication information $aut = (\widetilde{c}_j)_{j\in[t_p]}$ based on the algorithm Check. Concretely, set $\widetilde{c}_j = \prod_{i=1}^{t_p}(g^{v_i}/\prod_{k=1}^{\ell} c_k^{i^{t_p+k-1}})^{\gamma_{j,i}}$ for $j \in [t_p]$, where the $t_p \times t_p$ matrix $(\gamma_{j,i})_{1\leq j,i\leq t_p}$ is the inverse of the Vandermonde matrix $\mathbf{V}(1, \ldots, t_p)$:
3. Output $((v_i)_{i\in[t_p]}, aut)$.

It is direct that the outputs of simulator $\mathsf{Sim}(\mathbf{c}, I)$ are distributed identically to the outputs of algorithm $\mathsf{Share}^*(\mathbf{s})$ for participants in I.

A.3 A Proof for Theorem 4

Proof. We separately argue the three properties.

Acceptance. For all honestly generated $v_i = (s_i, r_i)$, c and $aut = (c_j)_{0\geq j\geq t_p-1}$, it holds that $g^{s_i}h^{r_i} = g^{\sum_{j=0}^{t_p} a_j\cdot i^j} h^{\sum_{j=0}^{t_p} b_j\cdot i^j} = \prod_{j=0}^{t_p} g^{a_j\cdot i^j} h^{b_j\cdot i^j} = \prod_{j=0}^{t_p-1} c_j^{i^j} \cdot c^{i^{t_p}}$.

$(t_p + 1)$-Correctness. For any set of participants $I \subset [n]$ where $|I| = m \geq t_p + 1$ and corresponding shares $(v_i)_{i\in I}$ where $v_i = (s_i, r_i)$, if it holds that $\mathsf{Check}(i, v_i, \mathbf{c}, aut) = 1$ for all $i \in I$, then we have $g^{s_i}h^{r_i} = \prod_{j=0}^{t_p-1} c_j^{i^j} \cdot c^{i^{t_p}}$ for all $i \in I$. Let $A(x) = \sum_{j=0}^{m-1} a_j \cdot x^j$ and $B(x) = \sum_{j=0}^{m-1} b_j \cdot x^j$ be the two polynomials that interpolate two sets of points $(i, s_i)_{i\in I}$ and $(i, r_i)_{i\in I}$ respectively. Note that, both the degrees of $A(x)$ and $B(x)$ are at most $m-1$. Then, it holds that $a_{t_p} = \sum_{i=1}^{m} \delta_{t_p+1,i} \cdot s_i \mod p$ and $b_{t_p} = \sum_{i=1}^{m} \delta_{t_p+1,i} \cdot r_i \mod p$, where the vector $(\delta_{t_p+1,i})_{i\in[m]}$ is the $(t_p + 1)$-th row of the matrix $\mathbf{V}(1, \ldots, m)^{-1}$. Therefore, we have

$$\begin{aligned} g^{a_{t_p}}h^{b_{t_p}} =& g^{\sum_{i=1}^{m}\delta_{t_p+1,i}\cdot s_i} h^{\sum_{i=1}^{m}\delta_{t_p+1,i}\cdot r_i} \\ =& \prod_{i=1}^{m} (g^{s_i}h^{r_i})^{\delta_{t_p+1,i}} \\ =& \prod_{i=1}^{m} (\prod_{j=0}^{t_p-1} c_j^{i^j} \cdot c^{i^{t_p}})^{\delta_{t_p+1,i}} \\ =& \prod_{j=0}^{t_p-1} c_j^{\sum_{i=1}^{m} i^j\cdot\delta_{t_p+1,i}} \cdot c^{\sum_{i=1}^{m} i^{t_p}\cdot\delta_{t_p+1,i}} \\ =& c \end{aligned}$$

t_p-privacy. We argue this by constructing the simulator $\mathsf{Sim}(c, I)$. Without loss of generality, we assume $I = \{1, \ldots, t_p\}$. On input a commitment c and a set I, the simulator $\mathsf{Sim}(c, I)$ proceeds as follows:

1. Simulate the private shares for participants in I, by picking $s_1, \ldots, s_{t_p}$, $r_1, \ldots, r_{t_p} \xleftarrow{R} \mathbb{Z}_p$ and setting $v_i = (s_i, r_i)$;
2. Compute the authentication information $aut = (c_j)_{0\geq j\geq t_p}$ based on the algorithm Check. Concretely, set $c_j = \prod_{i=1}^{t_p}(g^{s_i} \cdot h^{r_i}/c^{i^{t_p}})^{\gamma_{j,i}}$, where the $t_p \times t_p$ matrix $(\gamma_{j,i})_{1\leq j+1,i\leq t_p}$ is the inverse of Vandermonde matrix $\mathbf{v}(1, \ldots, t_p)$.
3. Output $((v_i)_{i\in I}, aut)$.

It is direct that outputs of simulator $\mathsf{Sim}(c, I)$ are distributed identically to the outputs of algorithm $\mathsf{Share}^*(s, r)$ for $(P_i)_{i\in I}$.

A.4 A Proof for Theorem 5

Proof. We separately argue the three properties.

Acceptance. For all honestly generated v_i, c and aut, it always holds that $v_i^e = (a \cdot s^i)^e = a^e \cdot (s^e)^i = aut \cdot c^i \mod N$.

2-Correctness. For any set of participants $I = \{i_j\}_{j\in|I|} \subset [n]$ where $|I| \geq 2$ and corresponding shares $(v_i)_{i\in I}$, if it holds that $\mathsf{Check}(i, v_i, c, aut) = 1$ for $i \in I$, then we have $v_i^e = aut \cdot c^i$ for $i \in I$. Therefore, the secret s output by Rec holds that $s^e = (c^\alpha(v_{i_2}/v_{i_1})^\beta)^e = c^{\alpha\cdot e}c^{(i_2-i_1)\beta} = c$.
1-privacy. we argue this by constructing the simulator $\mathsf{Sim}(c, i)$, where $i \in [e]$. Concretely, the simulator $\mathsf{Sim}(c, i)$ proceeds as follows:

1. Simulate the share for P_i, by picking random $v_i \xleftarrow{\mathrm{R}} \mathbb{Z}_N^*$;
2. Compute the authentication information $aut = v_i^e \cdot c^{-i}$.
3. Output (v_i, aut).

It is straightforward to check that the (v_i, aut) output by the simulator $\mathsf{Sim}(c, i)$ satisfies $\mathsf{Check}(i, v_i, c, aut) = 1$ and it is distributed identically to the output of algorithm $\mathsf{Share}^*(s)$ for P_i.

A.5 A Proof for Theorem 8

Proof. We separately argue the three properties.

Acceptance. For honestly generated $\mathbf{c} = (c_j)_{j\in[\ell]}$, $(v_i)_{i\in[n]}$ and $aut = (\widetilde{c}_j)_{j\in[t_p]}$, we show $\mathsf{Check}(i, v_i, \mathbf{c}, aut) = 1$ for all $i \in [n]$. First note that for honestly generated polynomials $A(x) = \sum_{k=1}^{\ell+t_p-1} a_k \cdot x^{k-1}$ and $B(x) = \sum_{k=1}^{\ell+t_p-1} b_k \cdot x^{k-1}$, their coefficients a_k's and b_k's hold that $a_k = \sum_{j=1}^{\ell} \delta_{k,j} \cdot s_j + \sum_{j=\ell+1}^{\ell+t_p} \delta_{k,j} \cdot \alpha_{j-\ell} \mod p$ and $b_k = \sum_{j=1}^{\ell} \delta_{k,j} \cdot r_j + \sum_{j=\ell+1}^{\ell+t_p} \delta_{k,j} \cdot \beta_{j-\ell} \mod p$, where the matrix $(\delta_{k,j})_{1\leq k,j\leq \ell+t_p}$ is equal to $\mathbf{V}(\boldsymbol{\zeta})^{-1}$. Moreover, if $\mathbf{c}$ and aut are computed honestly, we have $g^{a_k}h^{b_k} = \prod_{j=1}^{\ell} c_j^{\delta_{k,j}} \cdot \prod_{j=\ell+1}^{\ell+t_p} \widetilde{c}_{j-\ell}^{\delta_{k,j}} = h_k$ for all $k \in [\ell + t_p]$. Thus, for honestly generated v_i, parsed as (v_{i1}, v_{i2}), it holds that

$$\begin{aligned} g^{v_{i1}}h^{v_{i2}} &= g^{A(\eta_i)}h^{B(\eta_i)} \\ &= g^{\sum_{k=1}^{\ell+t_p} a_k\cdot\eta_i^{k-1}}h^{\sum_{k=1}^{\ell+t_p} b_k\cdot\eta_i^{k-1}} \\ &= \prod_{k=1}^{\ell+t_p}(g^{a_k}h^{b_k})^{\eta_i^{k-1}} \\ &= \prod_{k=1}^{\ell+t_p} h_k^{\eta_i^{k-1}}. \end{aligned}$$

$(\boldsymbol{\ell} + \boldsymbol{t_p})$-Correctness. For any set of participants $I \subset [n]$ where $|I| = m \geq t_p + \ell$ and corresponding shares $(v_i)_{i\in I}$ where $v_i = (v_{i1}, v_{i2})$, if it holds that $\mathsf{Check}(i, v_i, c, aut) = 1$ for all $i \in I$, then we have $g^{v_{i1}}h^{v_{i2}} = \prod_{k=1}^{\ell+t_p} h_k^{\eta_i^{k-1}}$ and $h_k = \prod_{j=1}^{\ell} c_j^{\delta_{k,j}} \cdot \prod_{j=\ell+1}^{\ell+t_p} \widetilde{c}_{j-\ell}^{\delta_{k,j}}$, where the matrix $(\delta_{k,j})_{1\leq k,j\leq\ell+t_p}$ is equal to $\mathbf{V}(\boldsymbol{\zeta})^{-1}$.

Let $A(x) = \sum_{k=1}^{m} a_k \cdot x^{k-1}$ and $B(x) = \sum_{k=1}^{m} b_k \cdot x^{k-1}$ be the two polynomials that interpolate the points $(\eta_i, v_{i1})_{i\in I}$ and $(\eta_i, v_{i2})_{i\in I}$, respectively. Note that, both the degrees of $A(x)$ and $B(x)$ are at most $m - 1$. Then, for all $k \in [m]$, it holds that $a_k = \sum_{i=1}^{m} \gamma_{k,i} \cdot v_{i1} \mod p$ and $b_k = \sum_{i=1}^{m} \gamma_{k,i} \cdot v_{i2} \mod p$, where the matrix $(\gamma_{k,i})_{1\leq k,j\leq m}$ is equal to $\mathbf{V}((\eta_i)_{i\in I})^{-1}$. Therefore, for all $j \in [\ell]$, we have

$$\begin{aligned} g^{s_j}h^{r_j} =& g^{A(\zeta_j)}h^{B(\zeta_j)} \\ =& g^{\sum_{k=1}^{m} a_k\cdot\zeta_j^{k-1}} \cdot h^{\sum_{k=1}^{m} b_k\cdot\zeta_j^{k-1}} \\ =& \prod_{k=1}^{m}\left(g^{a_k}\cdot h^{b_k}\right)^{\zeta_j^{k-1}} \\ =& \prod_{k=1}^{m}\left(g^{\sum_{i=1}^{m}\gamma_{k,i}\cdot v_{i1}} \cdot h^{\sum_{i=1}^{m}\gamma_{k,i}\cdot v_{i2}}\right)^{\zeta_j^{k-1}} \end{aligned}$$

$$
\begin{aligned}
&=\prod_{k=1}^{m}\left(\prod_{i=1}^{m}(g^{v_{i1}}h^{v_{i2}})^{\gamma_{k,i}}\right)^{\zeta_j^{k-1}}\\
&=\prod_{k=1}^{m}\left(\prod_{t=1}^{\ell+t_p}\left(h_t^{\sum_{i=1}^{m}\eta_i^{t-1}\cdot\gamma_{k,i}}\right)\right)^{\zeta_j^{k-1}}\\
&=\prod_{k=1}^{m}(h_k)^{\zeta_j^{k-1}}\\
&=\prod_{k=1}^{m}\left(\prod_{u=1}^{\ell}c_u^{\delta_{k,u}}\cdot\prod_{u=\ell+1}^{\ell+t_p}\widetilde{c}_{u-\ell}^{\delta_{k,u}}\right)^{\zeta_j^{k-1}}\\
&=\prod_{u=1}^{\ell}\left(c_u^{\sum_{k=1}^{m}\delta_{k,u}\cdot\zeta_j^{k-1}}\right)\cdot\prod_{u=\ell+1}^{\ell+t_p}\left(\widetilde{c}_{u-\ell}^{\sum_{k=1}^{m}\delta_{k,u}\cdot\zeta_j^{k-1}}\right)\\
&=c_j
\end{aligned}
$$

***t_p*-privacy**. we argue this by constructing a PPT simulator $\mathsf{Sim}((c_j)_{j\in\ell}, I)$ as below. Without loss of generality, we assume $I=\{1,\dots,t_p\}$. On input the commitments $(c_j)_{j\in\ell}\in\mathbb{G}^\ell$ and the set I, the simulator Sim proceeds as follows:

1. Simulate the private shares for participants in I, by picking two random vectors $(v_{i1})_{i\in[t_p]}, (v_{i2})_{i\in[t_p]}\xleftarrow{\text{R}}\mathbb{F}_p^{t_p}$ and setting $v_i=(v_{i1},v_{i2})$;
2. Compute the $t_p\times(\ell+t_p)$ matrix $\mathbf{G}$ as below:

$$
\mathbf{G}=\begin{bmatrix}1 & \eta_1 & \cdots & \eta_1^{\ell+t_p-1}\\ 1 & \eta_2 & \cdots & \eta_2^{\ell+t_p-1}\\ \vdots & \vdots & \ddots & \vdots\\ 1 & \eta_{t_p} & \cdots & \eta_{t_p}^{\ell+t_p-1}\end{bmatrix}\cdot\mathbf{V}(\boldsymbol{\zeta})^{-1},
$$

   Let $\mathbf{G}_1$ be the matrix formed by the first ℓ columns of $\mathbf{G}$ and $\mathbf{G}_2$ be the matrix formed by the last t_p columns of $\mathbf{G}$.
3. Compute $aut=(\widetilde{c}_1,\dots,\widetilde{c}_{t_p})$ on the basis of deterministic algorithm Check. Specifically, set $\widetilde{c}_k=\prod_{i=1}^{t_p}(g^{v_{i1}}h^{v_{i2}}/\prod_{j=1}^{\ell}c_j^{\gamma_{i,j}})^{v_{k,i}}$, where the $t_p\times\ell$ matrix $(\gamma_{i,j})_{i\in[t_p],j\in[\ell]}$ is equal to the matrix $\mathbf{G}_1$ and the $t_p\times t_p$ matrix $(v_{k,i})_{k,i\in[t_p]}$ is the inverse of the matrix $\mathbf{G}_2$:
4. Output $(v_1,\dots,v_{t_p},aut)$.

It is direct that the outputs of simulator Sim are distributed identically to the outputs of algorithm $\mathsf{Share}^*(\mathbf{s},\mathbf{r})$ for participants in I.