Non-Interactive Zero-Knowledge Proofs to Multiple Verifiers

Kang Yang[1][0000−0002−7453−4043] and Xiao Wang[2][0000−0002−5991−7417]

[1] State Key Laboratory of Cryptology, Beijing, China
yangk@sklc.org
[2] Northwestern University, Evanston, USA
wangxiao@cs.northwestern.edu

Abstract. In this paper, we study zero-knowledge (ZK) proofs for circuit satisfiability that can prove to n verifiers at a time efficiently. The proofs are secure against the collusion of a prover and a subset of t verifiers. We refer to such ZK proofs as multi-verifier zero-knowledge (MVZK) proofs and focus on the case that a majority of verifiers are honest (i.e., $t < n/2$). We construct efficient MVZK protocols in the random oracle model where the prover sends one message to each verifier, while the verifiers only exchange one round of messages. When the threshold of corrupted verifiers $t < n/2$, the prover sends $1/2 + o(1)$ field elements per multiplication gate to every verifier; when $t < n(1/2 - \epsilon)$ for some constant $0 < \epsilon < 1/2$, we can further reduce the communication to $O(1/n)$ field elements per multiplication gate per verifier. Our MVZK protocols demonstrate particularly high scalability: the proofs are streamable and only require a memory proportional to what is needed to evaluate the circuit in the clear.

1 Introduction

Zero-knowledge (ZK) proofs allow a prover $\mathcal{P}$, who knows a witness w, to convince a verifier $\mathcal{V}$ that $C(w) = 0$ for a circuit C, in the way that $\mathcal{V}$ learns nothing beyond the validity of the statement. One important type of ZK proofs is non-interactive ZK (NIZK), where the prover just needs to send one message to a verifier. This is particularly useful as the prover's message (i.e., the proof) can be reused to convince multiple verifiers. The efficiency of NIZK proofs has been significantly improved in recent years, based on different frameworks (e.g., [44,35,42,34,12,15,17,49,11,10,48,18,54,3,13] and references therein). Another important type of ZK proofs is designated-verifier ZK (DVZK), where an interactive protocol needs to be executed between the prover and the verifier. Compared to NIZK, DVZK protocols can often achieve a higher efficiency to prove to one verifier and scale to a very large circuit with a small memory. For example, recent DVZK proof systems [50,28,8,51,6,27] can prove tens of millions of gates per second with very limited bandwidth. However, such an advantage diminishes when the number of verifiers increases: DVZK protocols require the prover to execute the protocol with every verifier, while an NIZK proof enables all verifiers to verify the proof concurrently after the prover generates and publishes the proof.

In this work, we explore the middle ground between NIZK and DVZK: we study the efficiency of ZK proofs when a prover wants to prove to multiple verifiers (i.e., multi-verifier ZK, MVZK in short). This setting was first studied by Abe, Cramer and Fehr [2]. Specifically, we consider that a prover $\mathcal{P}$ needs to convince n verifiers $\mathcal{V}_1, \ldots, \mathcal{V}_n$, and the adversary can potentially corrupt a subset of t verifiers *and* optionally the prover. More specifically, we focus on the honest-majority setting, meaning that $t < n/2$ verifiers could be corrupted and can *collude* with the prover. Such an MVZK protocol is closely connected to DVZK in which the prover can only prove to a designated set of verifiers who are known ahead of the protocol execution. However, due to the fact that there is a majority of honest verifiers, it turns out the MVZK protocol can achieve some surprising features, e.g., being *non-interactive* between the prover and the verifiers in the information-theoretic setting.

Because of the involvement of multiple verifiers, there are two types of communications: 1) between the prover and verifiers and 2) between different verifiers. We say that the protocol is a *non-interactive* multi-verifier ZK (NIMVZK) proof if the prover only sends one message to each verifier. We say that the protocol is a *strong* NIMVZK proof if it is an NIMVZK and that there is only one round of communication between verifiers. We allow the verifiers to communicate for one round because without any communication between the verifiers, constructing NIMVZKs appears as difficult as constructing NIZKs.

In the MVZK setting, the known protocol [2] or those that can be implicitly constructed from the known techniques [14,16] either are *not* concretely efficient or *only* prove some specific circuits instead of generic circuits. Furthermore, none of the prior work considers how to stream the MVZK proofs, which is a crucial property to prove large-scale circuits.

1.1 Our Contribution

In this paper, we propose streamable NIMVZK protocols on generic circuits with both theoretical insights and practical implication. The protocols work in the honest-majority setting, meaning that the number t of corrupted verifiers is less than $n/2$, where n is the total number of verifiers. Compared to NIZKs, our NIMVZK protocols are much cheaper in terms of computational cost and use significantly less memory. Compared to DVZK, our protocols have three advantages: 1) the computation is still cheaper; 2) we can achieve the strongly non-interactive property; and 3) the communication is lower, especially when the number of verifiers is large. Specifically, our results are summarized as follows:

1. We present an information-theoretic NIMVZK protocol, where the prover sends $1 + o(1)$ field elements per multiplication gate to every verifier in one message (thus non-interactive), and the verifiers interact in both communication and rounds *logarithmic* to the circuit size. We consider the protocol as a stepping stone to introduce the following two main NIMVZK protocols.

2. Assuming a random oracle (and thus in the computational setting), we construct a strong NIMVZK proof based on Shamir secret sharing, where the verifiers only need to communicate for one round. The prover needs to send $1/2 + o(1)$ field elements per multiplication gate to each verifier and the communication cost between verifiers is still logarithmic.

   The challenge is that the message sent by the prover consists of the shares of every verifier that are private information and thus cannot be revealed. This makes the verifiers have no way to compute the public message that can be used as the input of a random oracle in the Fiat-Shamir transform. We proposed an efficient approach to allow Fiat-Shamir to work across multiple verifiers, i.e., enabling the prover to generate a *small* public message that can be *securely* used in the Fiat-Shamir transform, even if a minority of verifiers collude with the malicious prover.

3. When the corruption threshold is smaller (i.e., $t < n(1/2 - \epsilon)$ for some constant $0 < \epsilon < 1/2$), we use packed secret sharing (PSS) [30] to construct a strong NIMVZK protocol for proving a single generic circuit, which further reduces the communication complexity to $O(1/n)$ field elements per multiplication gate per verifier, while the communication complexity between verifiers is logarithmic to the circuit size. If applying the state-of-art secure multi-party computation (MPC) protocol [39] based on PSS to design an interactive MVZK protocol, the resulting protocol can achieve the same communication complexity. However, the constant in the O notation is significantly larger than our protocol.

   For a single generic circuit, PSS has been used in MPC protocols [24,32,31,39] in the honest-majority setting, but the overhead is often high due to the constraint how to pack the wire values to realize secure evaluation of the circuit. In the ZK setting, our strong NIMVZK protocol can remove the constraint, and achieve optimality for packing wire values and significantly better efficiency for checking correctness of packed sharings. For example, the state-of-the-art PSS-based MPC protocol [39] incurs a total communication cost of $O(n^5k^2)$ to check the consistency between packed input sharings and output sharings, where $k > \epsilon n + 1/2$ is the number of secrets packed in a single sharing. When being improved in the ZK setting, the total communication cost of our protocol can be reduced to $O(n^2k^2)$. Furthermore, we develop a non-interactive verification technique for checking correctness of PSS-based multiplication tuples, while the approach used in MPC [39] requires logarithmic rounds.

In summary, we designed concretely efficient MVZK protocols, which provide the attractive properties of NIZK (non-interactivity) and DVZK (memory efficiency and prover-computation efficiency). Although it is not applicable to all settings (due to the assumption of honest-majority verifiers), when it is applicable, the performance improvements to existing protocols are huge.

Streamable property of our NIMVZK. Although the communication complexity between the prover and verifiers is linear to the circuit size, all our protocols as described above are *streamable*, meaning that the prover can generate and send the proof on-the-fly and no party needs to store the whole proof during the protocol execution. As a result, the memory consumption of our protocols is proportional to what is needed to evaluate the statement in the clear. Furthermore, we make our strong NIMVZK proofs streamable in the way that the rounds among verifiers keep *unchanged* (i.e., only one round between verifiers is needed for proving multiple batches of gates).

Asymmetric property of our strong NIMVZK. One surprising feature of our two strong NIMVZK protocols in the computational setting is the *asymmetry* among verifiers. Specifically, among all verifiers, a subset of t verifiers only has a *sublinear* communication complexity: each verifier only receives $O(n + \log |C|)$ field elements from the prover, and only needs to send $O(n)$ field elements to other verifiers, where $|C|$ is the number of multiplication gates in a circuit C. It makes the protocols particularly suitable for the applications where the verifiers are a mix of powerful servers and lower-resource mobile devices.

1.2 Applications

Non-interactive MVZK proofs have the following applications:

1. **Drop-in replacement to NIZK and DVZK.** NIMVZK can be used in normal ZK applications as long as the identifies of the verifiers are known ahead of time and satisfy the security requirement (e.g., a majority of verifiers are honest for our NIMVZK protocols). For example, as described in [21], a ZK proof could potentially be used by Apple for auditing their Child Sexual Abuse Material detection protocol. Our NIMVZK protocol can be used when such an auditing needs to be performed to multiple agencies efficiently.
2. **Honest-majority MPC with input predicate check.** In some computational tasks, it is desired to execute the MPC protocol among multiple parties only if the input of every party is valid, where the validity is defined by some predicate. Although generic MPC can realize this functionality, using our NIMVZK protocols could further reduce the overhead of proving the predicate. As our protocols are based on Shamir sharings, it can be seamlessly integrated with MPC protocols also based on Shamir sharings.
3. **Private aggregation systems.** Systems like Prio [23] use a set of servers to collect and aggregate users' data. To prevent mistakes and attacks, users need to prove to the servers that their data is valid, which was done via *secret-shared non-interactive proof* in Prio. However, the protocol assumes the prover not to collude with any verifier for soundness. Our protocol could be a more efficient alternative and is sound even when a user colludes with a minority of servers. On the other hand, for zero-knowledge, Prio can tolerate all-but-one corrupted servers, while our protocols need to assume an honest majority of servers.

1.3 Related Work

The concept of multi-verifier ZK proofs was first discussed by Burmester and Desmedt [19], where they focus on how to save broadcasts. Lepinski, Micali and shelat [45] proposed a fair ZK proof, which ensures that even malicious verifiers who collude with the prover can learn nothing beyond the validity of the statement if the honest verifiers accept the proof. More recently, Baldimtsi et al. [5] proposed a crowd verifiable zero-knowledge proof, where the focus is to transform a Sigma protocol to their setting. All of the above works focus on extending the ZK functionality rather than the concrete efficiency of ZK proofs.

Abe, Cramer and Fehr [2] first studied the MVZK setting, and proposed a strong NIMVZK protocol for circuit satisfiability if at most $t < n/3$ verifiers are corrupted. Their protocol builds on the technique [1], and adopts the Pedersen commitment and verifiable secret sharing. Due to the usage of public-key operations for every non-linear gate, their protocol is *not* concretely efficient. In addition, the ZK proof by Groth and Ostrovsky [43] could be transformed into a strong NIMVZK proof in the corruption threshold of $t < n/2$, but their proof requires public-key operations per gate and thus is *not* concretely efficient. Compared to the NIMVZK proofs [2,43], our proofs do not require any public-key operation and are concretely efficient.

Although Boneh et al. [14] did *not* explicitly consider the MVZK setting, the ZK proofs proposed by them work in this setting. However, these protocols are only efficient applicable for circuits that can be represented by low-degree polynomials (instead of generic circuits). Recently, Boyle et al. [16] shown how to use the ZK proof [14] to design honest-majority MPC protocols with malicious security. However, they only considered how to prove correctness of degree-2 relations, and did *not* involve MVZK proofs on generic circuits yet. In addition, Boyle et al. [16] proposed an approach based on Fiat-Shamir to make the ZK proof on inner-product tuples non-interactive, where the difference between the secret and randomness needs to be sent. One can *generalize* their approach into our MVZK framework, and make the resulting MVZK proof strongly non-interactive. However, their approach requires $3\times$ larger communication than ours. Both works [14,16] did *not* consider how to make the ZK proofs *streamable*, which is addressed by our work.

One can also use maliciously secure MPC protocols in the honest-majority setting (e.g., [32,46,47,22,16,40,41,38,39]) to directly obtain interactive MVZK proofs. However, both of communication and computation costs will be significantly larger than our NIMVZK protocols. While NIZK can be transformed into NIMVZK directly, these NIZK proofs with performance similar to our proofs (e.g., recent succinct non-interactive proofs [48,18,54,13,53]) require memory linear to the circuit size, which could lead to a huge memory consumption for circuits with billions of gates. Our NIMVZK protocols are streamable, and the memory consumption of these protocols is proportional to what is needed to evaluate the circuit in the clear (meaning that these protocols only need a small memory cost for proving very large circuits). Compared to MVZK that is constructed from the recent VOLE-based DVZK proofs [50,28,8,51,6,27] by executing the protocol with every verifier, our strong NIMVZK proofs reduce round complexity from $O(1)$ to only one round between the prover and verifiers, and significantly improve efficiency.

Very recently, two works by Applebaum et al. [4] and Baum et al. [7] also presented MVZK protocols in the setting that a majority of verifiers are honest. Applebaum et al. [4] focuses on a theoretical perspective, and gave two strong NIMVZK protocols based on "Minicrypt"-type assumptions in the plain model. Baum et al. [7] adopted an approach similar to ours, and aim to construct concretely efficient MVZK protocols. In particular, they proposed two NIMVZK protocols that allow to identify the cheating verifiers (and thus have stronger security than ours); however, their protocols only tolerate a smaller number of corrupted verifiers (either $t < n/3$ or $t < n/4$). Neither of the works [4,7] adopted packed secret sharings and achieve the communication complexity of $O(1/n)$ per multiplication gate per verifier.

2 Preliminaries

We discuss some important preliminaries here and provide more preliminaries (e.g., security model) in the full version [52].

Notation. We use λ and ρ to denote the computational and statistical security parameters, respectively. We use $x \leftarrow S$ to denote that sampling x uniformly at random from a finite set S. For $a, b \in \mathbb{Z}$ with $a \leq b$, we write $[a, b] = \{a, \ldots, b\}$. We will use bold lower-case letters like $\boldsymbol{x}$ for column vectors, and denote by x_i the i-th component of $\boldsymbol{x}$ with x_1 the first entry. For two vectors $\boldsymbol{x}, \boldsymbol{y}$ of dimension m, $\boldsymbol{x} \odot \boldsymbol{y}$ denotes the inner product of $\boldsymbol{x}$ and $\boldsymbol{y}$ (i.e., $\boldsymbol{x} \odot \boldsymbol{y} = \sum_{i \in [1,m]} x_i \cdot y_i$). Sometimes, when the dimension of vectors $\boldsymbol{x}, \boldsymbol{y}$ is 1 (i.e., $\boldsymbol{x} = x$ and $\boldsymbol{y} = y$), we abuse the notation $\boldsymbol{x} \odot \boldsymbol{y}$ to denote the multiplication $x \cdot y$ for the sake of simplicity. We use $\log_k$ to denote the logarithm in base k, and denote by log the logarithm notation $\log_2$ for simplicity. For a finite field $\mathbb{F}$, we use $\mathbb{K}$ to denote a degree-r extension field of $\mathbb{F}$. In particular, we fix some monic, irreducible polynomial $f(X)$ of degree r and write $\mathbb{K} \cong \mathbb{F}[X]/f(X)$. Every field element $w \in \mathbb{K}$ can be denoted uniquely as $w = \sum_{h \in [1,r]} w_h \cdot X^{h-1}$ with $w_h \in \mathbb{F}$ for all $h \in [1, r]$. When we write arithmetic expressions involving both elements of $\mathbb{F}$ and elements of $\mathbb{K}$, it is understood that field elements in $\mathbb{F}$ are viewed as the polynomials lying in $\mathbb{K}$ that have only constant terms. For a circuit C, we use $|C|$ to denote the number of multiplication gates.

Zero-knowledge functionality. Our ZK functionality for proving circuit satisfiability against multiple verifiers is shown in Figure 1. Let n be the total number of verifiers. We consider the MVZK protocols in the honest-majority setting, i.e, the adversary allows to corrupt at most $t < n/2$ verifiers. The adversary is also allowed to corrupt the prover. When the prover is honest, functionality $\mathcal{F}_{\mathsf{mvzk}}$ defined in Figure 1 captures *zero-knowledge*, meaning that t malicious verifiers cannot learn any information on the witness. When the prover is malicious, $\mathcal{F}_{\mathsf{mvzk}}$

Functionality $\mathcal{F}_{\mathsf{mvzk}}$

This functionality runs with a prover $\mathcal{P}$ and n verifiers $\mathcal{V}_1, \ldots, \mathcal{V}_n$. Let $\mathcal{H}$ denote the set of honest verifiers. This functionality operates as follows:

1. Upon receiving $(\mathsf{prove}, C, \boldsymbol{w})$ from $\mathcal{P}$ and (verify, C) from $\mathcal{V}_i$ for all $i \in [1, n]$ where C is a circuit, set $b := \mathsf{true}$ if $C(\boldsymbol{w}) = 0$ and $b := \mathsf{false}$ otherwise.
2. Send b to the adversary. For each $i \in \mathcal{H}$, wait for an input from the adversary, and then do the following:
   - If it is $\mathsf{continue}_i$, send b to the verifier $\mathcal{V}_i$.
   - If it is abort_i, send abort to the verifier $\mathcal{V}_i$.

Fig. 1: **Zero-knowledge functionality for honest-majority verifiers.**

captures *soundness*, i.e., the malicious prover cannot make the honest verifiers accept if $C(\boldsymbol{w}) \neq 0$, even though it *colludes* with t malicious verifiers.

We can consider MVZK protocols as special MPC protocols. Thus, we adopt the notion of *security with abort* in the MPC setting to define $\mathcal{F}_{\mathsf{mvzk}}$ and other functionalities defined in the subsequent sections, where the corrupted verifiers may receive output while the honest verifiers do not. Our definition does not guarantee *unanimous abort*, meaning that some honest verifiers may receive output while other honest verifiers abort. Nevertheless, it is easy to tune our protocols to satisfy the security notion of *unanimous abort*, by having the verifiers broadcast whether they will abort or not at the end of the protocol execution [36].

Communication model. The default communication between the prover and verifiers is private channel, unless otherwise specified. We assume that all verifiers are connected via authenticated channels. In the computational setting, the prover sometimes needs to communicate with all verifiers over a broadcast channel. Since we allow abort, the broadcast channel can be established using a standard echo-broadcast protocol [36], where the communication overhead can be improved to be constant small using a collision-resistant hash function. In our strong NIMVZK protocols, the verifiers need to exchange the shares in one round at the end of protocol execution. In parallel with the communication of shares, every verifier can send the hash output of the messages broadcast by the prover to all other verifiers. Therefore, although the echo-broadcast protocol is used in our MVZK proofs, we can still achieve strongly non-interactive.

Linear secret sharing scheme. In our NIMVZK protocols, we will extensively use *linear secret sharing schemes* (LSSSs) with a threshold t. A t-out-of-n LSSS enables a secret x to be shared among n parties, such that no subset of t parties can learn any information on x, while any subset of $t+1$ parties can reconstruct the secret. To align with the description of our NIMVZK protocols, we let the prover $\mathcal{P}$ play the role of the dealer and let every verifier $\mathcal{V}_i$ obtain the shares. We require that LSSS supports the following procedures:

– $[x] \leftarrow \mathsf{Share}(x)$: In this procedure, a dealer $\mathcal{P}$ shares a secret x among the parties $\mathcal{V}_1, \ldots, \mathcal{V}_n$, such that $\mathcal{V}_i$ gets a share x^i for $i \in [1, n]$. The sharing of x output by this procedure is denoted by $[x]$.
– $x \leftarrow \mathsf{Open}([x])$: Given a sharing $[x]$, this procedure is executed by parties $\mathcal{V}_1, \ldots, \mathcal{V}_n$. At the end of the execution, if $[x]$ is not valid, then all honest parties abort; otherwise, every party will output x.
– *Linear combination*: Given the public coefficients $c_0, c_1, \ldots, c_\ell$ and secret sharings $[x_1], \ldots, [x_\ell]$, $\mathcal{V}_1, \ldots, \mathcal{V}_n$ can *locally* compute $[y] = \sum_{i=1}^{\ell} c_i \cdot [x_i] + c_0$, such that $y = \sum_{i=1}^{\ell} c_i \cdot x_i + c_0$ holds.

We describe two LSSS instantiations shown in the full version [52], where one is Shamir secret sharing and the other is packed secret sharing (a generalization of Shamir secret sharing). For Shamir secret sharing, for a vector $\boldsymbol{x} = (x_1, \ldots, x_m)$, we will use $[\boldsymbol{x}]$ to denote $([x_1], \ldots, [x_m])$. For packed secret sharing, for a vector $\boldsymbol{x} \in \mathbb{F}^k$, we will use $[\boldsymbol{x}]$ to denote a single packed sharing that stores k secrets of $\boldsymbol{x}$. We assume that the shares of any t parties are uniformly random, which is satisfied by the two instantiations.

3 Technical Overview

We describe the ideas in our NIMVZK protocols and how we come up with these constructions in this section. We leave the full details and their proofs of security in later sections.

3.1 Information-Theoretic Non-Interactive MVZK

We introduce our non-interactive MVZK proofs starting from an *information-theoretic* NIMVZK protocol that is a warm-up to describe the techniques in our strong NIMVZK protocols.

Our approach for NIMVZK. Our NIMVZK proofs follow the "commit-and-prove" paradigm, where secrets are committed using Shamir sharings and the security of commitments is guaranteed in the honest-majority setting. At a high level, our information-theoretic protocol (as well as other two protocols discussed later) have the following steps.

1. For the output z of each circuit-input gate or multiplication gate, the prover runs $\mathsf{Share}(z)$ to distribute the shares of $[z]$ to all verifiers. Since LSSS is used, the addition gates can be locally computed by the verifiers.
2. For a circuit with N multiplication gates, we have N multiplication triples $([x_i], [y_i], [z_i])$ over a field $\mathbb{F}$ that the verifiers need to check. All parties jointly sample a uniform element $\chi \in \mathbb{K}$, and then compute the inner-product tuple:

$$[\boldsymbol{x}] := \left([x_1], \ldots, \chi^{N-1} \cdot [x_N]\right), \ [\boldsymbol{y}] := ([y_1], \ldots, [y_N]), \ [z] := \textstyle\sum_{i=1}^{N} \chi^{i-1} \cdot [z_i].$$

If there exists one *incorrect* multiplication triple, then the inner-product tuple defined as above is also *incorrect*, except with probability $\frac{N-1}{|\mathbb{K}|}$.

3. The verifiers check correctness of the inner-product tuple $([\boldsymbol{x}], [\boldsymbol{y}], [z])$ with logarithmic communication.

In the information-theoretic setting, the verifiers can call a coin-tossing functionality (shown in the full version [52]) to sample the coefficient χ, but χ is *not* available to the prover while keeping non-interactive between the prover and verifiers. The distributed ZK proofs by Boneh et al. [14] could check correctness of an inner-product tuple, but it only works when the prover knows the secrets. To use their protocol directly, we would need the verifiers to send χ to the prover, and then the round complexity between the prover and verifiers will be at least 3 rounds. Our task is to design a non-interactive protocol that verifies correctness of an inner-product tuple, where the secrets are shared among verifiers and *unknown to the prover*. We adapt the checking approach by Goyal et al. [40,41] (building upon the technique [14]) from the MPC setting to the MVZK setting, and construct a verification protocol to check correctness of inner-product tuples. In particular, our verification protocol makes the prover generate the random sharings and random multiplication triples (instead of letting the verifiers run the DN multiplication protocol [25] that is done in [40,41]), which is sufficient for MVZK as zero-knowledge only needs to hold for an honest prover.

3.2 Distributing Fiat-Shamir for Strong Non-Interactive MVZK

With the above preparation, we now discuss how to construct a strong NIMVZK proof, where the verifiers communicate for only one round. This is a highly non-trivial task, as it is even unclear how to sample a random coefficient $\chi \in \mathbb{K}$ as needed in step 2. Since every verifier can only send one message to other verifiers, using a secure coin-tossing protocol is not possible. The other randomness source that we can use is random oracle (i.e., adopting the Fiat-Shamir heuristic). However, only the shares are sent by the prover where the shares need to be kept secret, and thus the verifiers has no way to compute a public message that can be used as the input of a random oracle. This was in fact attempted in the distributed ZK proof [14] as well, but their non-interactive solution does not allow the prover to collude with any verifier.

Let's first review how Boneh et al. [14] use Fiat-Shamir in the case that all verifiers do not collude with the prover. Suppose that the prover $\mathcal{P}$ sends a message Msg_i along with a randomness r_i to a verifier $\mathcal{V}_i$ for $i \in [1,n]$, where Msg_i and r_i need to be kept secret. Every verifier $\mathcal{V}_i$ can send $\nu_i := \mathsf{H}(\mathsf{Msg}_i, r_i)$ to other verifiers where H is a random oracle, and then generates a random challenge $\chi := \bigoplus_{i\in[1,n]} \nu_i$, when ignoring some details for simplicity. Prover $\mathcal{P}$ can also compute the challenge χ as it knows all messages and randomness. When the prover is corrupted (and thus we are concerning soundness), all verifiers are assumed to be honest and thus can exchange the correct values $\{\nu_i\}_{i\in[1,n]}$, so that the verifiers can compute a random challenge χ to execute the protocol. However, when $\mathcal{P}$ colludes with a verifier $\mathcal{V}_{i^*}$, this method does not work anymore: $\mathcal{P}$ can cheat when the challenge is some value $\chi* \neq \bigoplus_{i\in[1,n]} \mathsf{H}(\mathsf{Msg}_i, r_i)$; after receiving the values of other verifiers, $\mathcal{V}_{i^*}$ can compute $\nu_{i^*} = \mathsf{H}(\mathsf{Msg}_{i^*}, r_{i^*})$ and the correct challenge χ, and then send $\nu'_{i^*} = \nu_{i^*} \oplus \chi \oplus \chi*$ to every other verifier such that the invalid proof can still go through.

Because of the round-complexity requirement on the verifier side, we cannot let the verifiers to sample χ. So it appears that in order to get a strong NIMVZK, we must find an approach to enable the prover to generate public messages via some sort of Fiat-Shamir transformation in the distributed setting so that: 1) the protocol tolerates the collusion of the prover and a minority of verifiers, and 2) does not require the verifiers to interact more than one round. Let H, H' be two random oracles with exponentially large ranges. Our technique to support Fiat-Shamir is presented as follows.

1. Suppose that the prover $\mathcal{P}$ sends (Msg_i, r_i) to every verifier $\mathcal{V}_i$ over a private channel, where Msg_i consists of the shares held by $\mathcal{V}_i$ for our protocols.
2. Now, $\mathcal{P}$ also broadcasts commitments $com_i := \mathsf{H}(\mathsf{Msg}_i, r_i)$ for all $i \in [1, n]$ to all verifiers, where the broadcast does not increase the rounds between verifiers that has been explained in Section 2.
3. Every verifier $\mathcal{V}_i$ checks that $com_i = \mathsf{H}(\mathsf{Msg}_i, r_i)$. As we assume that $t < n/2$, we can guarantee that a majority of commitments in $com_1, \ldots, com_n$ are computed correctly.
4. The verifiers can generate a random challenge $\chi := \mathsf{H}'(com_1, \ldots, com_n)$, as they now know the public messages $com_1, \ldots, com_n$. Then, the verifiers use χ to transform the verification of N multiplication triples into that of an inner-product tuple as described in Section 3.1.

If H has a 2λ-bit output length and thus is collision-resistant, then it would make com_i binding and the proof can easily go through, where all the commitments $\{com_i\}$ held by $n - t$ honest verifiers will uniquely define the secrets on all wires. However, we make a key observation that it is sufficient to prove security, if the challenge χ is guaranteed to be defined after the secrets on all wires have been determined (i.e., χ is independent of these secrets). Therefore, it is unnecessary to require the collision resistance for H, but rather we only need H to be second preimage-resistant, which allows to achieve better efficiency, e.g., using the construction [26]. In particular, if χ has been defined and known by the malicious prover, then it must make a query $(com_1, \ldots, com_n)$ to random oracle H'. Then, the malicious prover cheats to find a pair (Msg'_i, r'_i) associated with χ, and then sends it to some honest verifier $\mathcal{V}_i$. The cheat will not be detected only if $com_i = \mathsf{H}(\mathsf{Msg}'_i, r'_i)$, which is equivalent to find either a preimage or a second preimage of com_i. The Fiat-Shamir approach as described above only introduces a small communication overhead, i.e., $O(n^2\lambda)$ bits in total between the prover and all verifiers that is independent of the circuit size.

Through the above approach, the verifiers can generate a random challenge non-interactively, and then use it to convert the verification of multiplication triples into that of an inner-product tuple. We can simplify the verification technique (shown in Section 3.3) by viewing Shamir secret sharing as a special case of packed secret sharing, and then use it to verify the inner-product tuple in one round between verifiers. The resulting strong NIMVZK protocol is streamable while keeping the round complexity between verifiers unchanged (see below).

3.3 More Efficient Strong NIMVZK from Packed Secret Sharing

The above discussion shows a strong NIMVZK protocol where a prover sends one message to each verifier and the verifiers communicate only one round. It is secure against the adversary corrupting up to a minority of verifiers (i.e., $t < n/2$) and the prover. However, the downside is the communication of $1/2 + o(1)$ field elements per multiplication gate per verifier, and a majority of the proof is used to transmit the shares of wire values. We now discuss the strong NIMVZK protocol that reduces the communication cost to $O(1/n)$ field elements per multiplication gate per verifier, when the threshold of corrupted verifiers $t < n(1/2 - \epsilon)$ for any $0 < \epsilon < 1/2$. This protocol adopts packed secret sharing (PSS) [30] as the underlying LSSS, where each sharing packs $k = O(n)$ secrets.

Using packed secret sharing efficiently for a single generic circuit is a huge challenge, because the layout of the circuit could be complicated for packing k gates, and it is not possible to move around any individual wire when using PSS. In fact, because of this, prior MPC works [37,9] using PSS focus on SIMD operations (i.e., repeated circuits). For a single generic circuit, the state-of-the-art PSS-based MPC protocol [39] requires to evaluate the circuit layer-by-layer that needs the rounds linear to the circuit depth, and splits each output wire into different output wires that each can be used only once. Fortunately, we observe that even a single generic circuit can be packed optimally in the context of zero-knowledge, and can remove the constraints in MPC. Particularly, the prover can prove a circuit in a streamable way without the constraint of proving the circuit layer-by-layer, as the prover knows all the wire values.

Consistency check of wire values. In our NIMVZK protocol, if the out degree of a gate is greater than 1, we allow an output wire to appear multiple times (instead of splitting the output wire into multiple output wires), which enables us to obtain better communication. In this case, we need to use the consistency check to ensure that the same wire is assigned with the same value. Specifically, for each input packed sharing $[\boldsymbol{y}]$, if the j-th secret y_j comes from the i-th secret x_i stored in an output packed sharing $[\boldsymbol{x}]$, then we need to check $x_i = y_j$. This corresponds to the wire that carries the value $x_i = y_j$. Following the work [39], we refer to $([\boldsymbol{x}], [\boldsymbol{y}], i, j)$ as a wire tuple. For the consistency check of wire tuples, we reduce the total communication complexity from $O(n^5 k^2)$ in MPC [39] to $O(n^2 k^2)$ for our strong NIMVZK protocol. For each $i, j \in [1, k]$, let $([\boldsymbol{x}_1], [\boldsymbol{y}_1], i, j), \ldots, ([\boldsymbol{x}_m], [\boldsymbol{y}_m], i, j)$ be the wire tuples with the same indices i, j. We use the random-linear-combination approach to check the consistency. Specifically, the prover $\mathcal{P}$ samples two random vectors $\boldsymbol{x}_0, \boldsymbol{y}_0$ such that $x_{0,i} = y_{0,j}$, and then distributes the shares of $[\boldsymbol{x}_0]$ and $[\boldsymbol{y}_0]$ to all verifiers. To support Fiat-Shamir, we need $\mathcal{P}$ to generate these shares in two steps: 1) distributing the shares of two random sharings $[\boldsymbol{r}]$ and $[\boldsymbol{s}]$; and 2) broadcasts the differences $\boldsymbol{u} = \boldsymbol{x}_0 + \boldsymbol{r}$ and $\boldsymbol{v} = \boldsymbol{y}_0 + \boldsymbol{s}$ to all verifiers. Then the verifiers can locally compute $[\boldsymbol{x}_0] := \boldsymbol{u} - [\boldsymbol{r}]$ and $[\boldsymbol{y}_0] := \boldsymbol{v} - [\boldsymbol{s}]$. $\mathcal{P}$ and all verifiers can generate a random challenge $\alpha = \mathsf{H}'(\chi, \boldsymbol{u}, \boldsymbol{v}, i, j)$, where χ is another random challenge related to the secrets $\{(\boldsymbol{x}_h, \boldsymbol{y}_h)\}_{h \in [1,m]}$. Then, the verifiers can now check correctness of the following wire tuple:

$$[\boldsymbol{x}] := \sum_{h=1}^{m} \alpha^h \cdot [\boldsymbol{x}_h] + [\boldsymbol{x}_0],\ [\boldsymbol{y}] := \sum_{h=1}^{m} \alpha^h \cdot [\boldsymbol{y}_h] + [\boldsymbol{y}_0].$$

This check can be done by letting the verifiers open $([\boldsymbol{x}], [\boldsymbol{y}])$ and check $x_i = y_j$. When streaming the strong NIMVZK protocol, the verification of wire tuples do *not* increase the rounds between verifiers (see Section 6.1 for details).

Verification of PSS-based inner-product tuples. Once we enable Fiat-Shamir as shown in Section 3.2, we also get another benefit that now the challenge χ is also known to the prover $\mathcal{P}$. Thus, we can non-interactively transform the verification of PSS-based multiplication tuples into that of a packed inner-product tuple. We present a *non-interactive* technique to verify the correctness of a packed inner-product tuple, which is inspired by prior work [14,16,40,41,39]. We also adapt the technique by Baum et al. [8] from the DVZK setting to the MVZK setting in order to further improve computational efficiency. Our verification approach has lower round complexity than that used in PSS-based MPC [39] (one round vs logarithm rounds). At a high level, our protocol for verifying correctness of a packed inner-product tuple works as follows:

1. Suppose that all verifiers hold the shares of a dimension-M packed inner-product tuple $(([\boldsymbol{x}_1], \ldots, [\boldsymbol{x}_M]), ([\boldsymbol{y}_1], \ldots, [\boldsymbol{y}_M]), [\boldsymbol{z}])$, where $\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i \in [1,M]}$ and $\boldsymbol{z}$ are secret vectors in $\mathbb{K}^k$. Prover $\mathcal{P}$ knows all the secret vectors, and wants to prove $\boldsymbol{z} = \sum_{i \in [1,M]} \boldsymbol{x}_i * \boldsymbol{y}_i$ where $*$ denotes the component-wise product.
2. The verifiers *recursively* reduce the dimension of $(([\boldsymbol{x}_1], \ldots, [\boldsymbol{x}_M]), ([\boldsymbol{y}_1], \ldots, [\boldsymbol{y}_M]), [\boldsymbol{z}])$ to 2. This is performed by splitting a packed inner-product tuple $(([\boldsymbol{x}_1], \ldots, [\boldsymbol{x}_m]), ([\boldsymbol{y}_1], \ldots, [\boldsymbol{y}_m]), [\boldsymbol{z}])$ into two inner-product tuples $(([\boldsymbol{a}_{1,1}], \ldots, [\boldsymbol{a}_{1,\ell}]), ([\boldsymbol{b}_{1,1}], \ldots, [\boldsymbol{b}_{1,\ell}]), [\boldsymbol{c}_1])$ and $(([\boldsymbol{a}_{2,1}], \ldots, [\boldsymbol{a}_{2,\ell}]), ([\boldsymbol{b}_{2,1}], \ldots, [\boldsymbol{b}_{2,\ell}]), [\boldsymbol{c}_2])$, where $\ell = m/2$ and $[\boldsymbol{z}] = [\boldsymbol{c}_1] + [\boldsymbol{c}_2]$. Then, we use a protocol to compress the two packed inner-product tuples into one inner-product tuple.
3. To realize the above splitting step, $\mathcal{P}$ can directly distribute the shares of $[\boldsymbol{c}_1] = [\sum_{h \in [1,\ell]} \boldsymbol{a}_{1,h} * \boldsymbol{b}_{1,h}]$ to all verifiers. However, this does not support Fiat-Shamir, as no public message is available. Instead, we let $\mathcal{P}$ distribute the shares of a random packed sharing $[\boldsymbol{r}]$, and then broadcast a public message $\boldsymbol{u} = \boldsymbol{c}_1 + \boldsymbol{r}$ to all verifiers, who can locally compute $[\boldsymbol{c}_1] := \boldsymbol{u} - [\boldsymbol{r}]$. Then, the verifiers can locally compute $[\boldsymbol{c}_2] := [\boldsymbol{z}] - [\boldsymbol{c}_1]$.
4. We adopt the polynomial approach to compress two packed inner-product tuples $(([\boldsymbol{a}_{1,1}], \ldots, [\boldsymbol{a}_{1,\ell}]), ([\boldsymbol{b}_{1,1}], \ldots, [\boldsymbol{b}_{1,\ell}]), [\boldsymbol{c}_1])$ and $(([\boldsymbol{a}_{2,1}], \ldots, [\boldsymbol{a}_{2,\ell}]), ([\boldsymbol{b}_{2,1}], \ldots, [\boldsymbol{b}_{2,\ell}]), [\boldsymbol{c}_2])$ into a single tuple $(([\boldsymbol{x}_1], \ldots, [\boldsymbol{x}_\ell]), ([\boldsymbol{y}_1], \ldots, [\boldsymbol{y}_\ell]), [\boldsymbol{z}])$, which has been used in prior work such as [39]. Differently, we will use the Fiat-Shamir transform to realize the non-interactive compression. Specifically, the parties compute the sharings of polynomials $[\boldsymbol{f}_j(\cdot)]$, $[\boldsymbol{g}_j(\cdot)]$ for $j \in [1, \ell]$ and $[\boldsymbol{h}(\cdot)]$, such that $\boldsymbol{f}_j(i) = \boldsymbol{a}_{i,j}$, $\boldsymbol{g}_j(i) = \boldsymbol{b}_{i,j}$ and $\boldsymbol{h}(i) = \boldsymbol{c}_i$ for $i \in [1, 2]$. Then $\mathcal{P}$ needs to convince the verifiers that $\boldsymbol{h}(X) = \sum_{j \in [1,\ell]} \boldsymbol{f}_j(X) * \boldsymbol{g}_j(X)$, which can be realized by proving $\boldsymbol{h}(\alpha) = \sum_{j \in [1,\ell]} \boldsymbol{f}_j(\alpha) * \boldsymbol{g}_j(\alpha)$ for a random challenge α. $\mathcal{P}$ and all verifiers can generate α by computing $\mathsf{H}'(\gamma, msg)$

where γ is the challenge used in the previous iteration and msg is the public message sent in the current iteration. Now, the parties can define $([\boldsymbol{x}_j] = [\boldsymbol{f}_j(\alpha)], [\boldsymbol{y}_j] = [\boldsymbol{g}_j(\alpha)])$ for $j \in [1, \ell]$ and $[\boldsymbol{z}] = [\boldsymbol{h}(\alpha)]$, and execute the next iteration.

5. Let $(([\boldsymbol{x}_1], [\boldsymbol{x}_2]), ([\boldsymbol{y}_1], [\boldsymbol{y}_2]), [\boldsymbol{z}])$ be the packed inner-product tuple after the dimension reduction was completed. We can adapt the randomization technique [14,39] to check the correctness of this tuple. In the same way, we can split it into two multiplication tuples $([\boldsymbol{x}_1], [\boldsymbol{y}_1], [\boldsymbol{z}_1])$ and $([\boldsymbol{x}_2], [\boldsymbol{y}_2], [\boldsymbol{z}_2])$ with $[\boldsymbol{z}] = [\boldsymbol{z}_1] + [\boldsymbol{z}_2]$. The prover $\mathcal{P}$ can distribute the shares of a random multiplication tuple $([\boldsymbol{x}_0], [\boldsymbol{y}_0], [\boldsymbol{z}_0])$ with $\boldsymbol{z}_0 = \boldsymbol{x}_0 * \boldsymbol{y}_0$ to all verifiers in the way compatible with Fiat-Shamir. Then, $\mathcal{P}$ and the verifiers can compress $\{([\boldsymbol{x}_i], [\boldsymbol{y}_i], [\boldsymbol{z}_i])\}_{i\in[0,2]}$ into $([\boldsymbol{x}], [\boldsymbol{y}], [\boldsymbol{z}])$. All verifiers can run the Open procedure to obtain $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$ and check that $\boldsymbol{z} = \boldsymbol{x} * \boldsymbol{y}$.

Streaming strong NIMVZK with the same round complexity. We can use the strong NIMVZK protocol to prove a very large circuit in a streamable way, such that between the prover and verifiers are non-interactive for proving a batch of $N = k \cdot M$ multiplication gates each time, and the verifiers *still* communicate only one round for proving the whole circuit. For a batch of $N = k \cdot M$ multiplication gates, the parties can transform M PSS-based multiplication tuples into a packed inner-product tuple with dimension M, and then compress it into a packed inner-product tuple denoted by $\mathsf{IPtuple}_1$ with dimension $M/2^c$ for some integer $c \geq 1$. For another batch of multiplication gates, the parties can generate another packed inner-product tuple $\mathsf{IPtuple}_2$ with dimension $M/2^c$ in the same way. Then, the prover and verifiers can compress $\mathsf{IPtuple}_1$ and$\mathsf{IPtuple}_2$ into a packed inner-product tuple $\mathsf{IPtuple}_3$ with the same dimension $M/2^c$, where the challenge α for this compression is computed with random oracle H' and two challenges to obtain $\mathsf{IPtuple}_1$ and$\mathsf{IPtuple}_2$. After the whole circuit has been evaluated, the verifiers can check correctness of the final packed inner-product tuple (with dimension $M/2^c$) stored in memory by communicating only one round. As a result, all parties only need memory linear to what is needed to evaluate the statement in the clear.

4 Information-Theoretic NIMVZK Proof

We present a non-interactive multi-verifier zero-knowledge (NIMVZK) protocol with information-theoretic security in the $(\mathcal{F}_{\mathsf{coin}}, \mathcal{F}_{\mathsf{verifyprod}})$-hybrid model, assuming an honest majority of verifiers, where $\mathcal{F}_{\mathsf{coin}}$ is a coin-tossing functionality shown in the full version [52]. Functionality $\mathcal{F}_{\mathsf{verifyprod}}$ allows to verify the correctness of an inner-product tuple secretly shared among verifiers. It is possible to instantiate $\mathcal{F}_{\mathsf{verifyprod}}$ using prior work on fully linear PCP (or IOP) [14], but we can improve its communication (or rounds) by adapting the technique by Goyal et al. [40,41] in the MPC setting to the MVZK setting.

Functionality $\mathcal{F}_{\mathsf{verifyprod}}$

This functionality runs with a prover $\mathcal{P}$ and n verifiers $\mathcal{V}_1, \ldots, \mathcal{V}_n$. Let $([\boldsymbol{x}], [\boldsymbol{y}], [z])$ be an inner-product tuple defined over a field $\mathbb{K}$, where the dimension of vectors $\boldsymbol{x}, \boldsymbol{y}$ is N. This functionality operates as follows:

1. Upon receiving the shares of $[\boldsymbol{x}], [\boldsymbol{y}], [z]$ from honest verifiers, run as follows:
   - Reconstruct the secret vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}^N$ and secret $z \in \mathbb{F}$ from these shares.
   - Compute the shares of $[\boldsymbol{x}], [\boldsymbol{y}], [z]$ held by corrupted verifiers, and send them to the adversary.
   - If $\mathcal{P}$ is corrupted, send $(\boldsymbol{x}, \boldsymbol{y}, z)$ to the adversary.
2. If $z \neq \boldsymbol{x} \odot \boldsymbol{y}$, then set $b := \mathsf{abort}$, otherwise set $b := \mathsf{accept}$. Send b to the adversary. For $i \in \mathcal{H}$, wait for an input from the adversary, and do the following:
   - If it is $\mathsf{continue}_i$, send b to $\mathcal{V}_i$.
   - If it is abort_i, send abort to $\mathcal{V}_i$.

Fig. 2: **Zero-knowledge verification functionality for an inner-product tuple.**

4.1 From General Adversaries to Maximal Adversaries for MVZK

Before we describe the NIMVZK protocol, we prove an important lemma that can be used to simplify the proofs of the MVZK protocols in this paper and the future works. Informally, this lemma states that if an MVZK protocol is secure against *exactly* t malicious verifiers, then the protocol is also secure against *at most* t malicious verifiers. The proof of this lemma is based on that of a similar lemma for honest-majority MPC by Genkin et al. [33]. This lemma allows us to only consider the maximum adversaries who corrupt exactly t verifiers, and thus simplifies the security proofs of MVZK protocols. One caveat is that the proof of this lemma needs to specially deal with the case that the honest verifiers will receive output as well as the possible random-oracle queries (e.g., the Fiat-Shamir transform [29] is used).

Lemma 1. *Let Π be an MVZK protocol proving the satisfiability of a circuit C for $n \geq 2t+1$ verifiers. Then, if protocol Π securely realizes $\mathcal{F}_{\mathsf{mvzk}}$ in the presence of any malicious adversary corrupting exactly t verifiers, then Π securely realizes $\mathcal{F}_{\mathsf{mvzk}}$ against any malicious adversary corrupting at most t verifiers.*

The proof of the above lemma is given in the full version [52]. The above lemma can be applied to not only our information-theoretic NIMVZK protocol but also the strong NIMVZK proofs in the computational setting that will be described in Section 5 and Section 6.

4.2 Our Information-Theoretic NIMVZK Protocol

In Figure 3, we describe the detailed NIMVZK protocol with information theoretic security in the $(\mathcal{F}_{\mathsf{coin}}, \mathcal{F}_{\mathsf{verifyprod}})$-hybrid model. For each circuit-input gate

Protocol $\Pi^{\mathsf{it}}_{\mathsf{nimvzk}}$

Inputs: A prover $\mathcal{P}$ holds a witness $\boldsymbol{w} \in \mathbb{F}^m$. $\mathcal{P}$ and all verifiers $\mathcal{V}_1, \ldots, \mathcal{V}_n$ hold an arithmetic circuit C over a field $\mathbb{F}$ with $|\mathbb{F}| > n$. Let N denote the number of multiplication gates in the circuit. $\mathcal{P}$ will convince the verifiers that $C(\boldsymbol{w}) = 0$.

Circuit evaluation: In a predetermined topological order, $\mathcal{P}$ and all verifiers evaluate the circuit as follows:

- For each circuit-input wire with input value $w \in \mathbb{F}$, $\mathcal{P}$ (acting as the dealer) runs $[w] \leftarrow \mathsf{Share}(w)$ to distribute the shares to all verifiers.
- For each addition gate with input sharings $[x]$ and $[y]$, all verifiers locally compute $[z] := [x] + [y]$, and $\mathcal{P}$ computes $z := x + y \in \mathbb{F}$.
- For each multiplication gate with input values $x, y \in \mathbb{F}$, $\mathcal{P}$ computes $z := x \cdot y \in \mathbb{F}$, and then executes $[z] \leftarrow \mathsf{Share}(z)$ which distributes the shares to all verifiers.

Verification of multiplication gates: Let $([x_i], [y_i], [z_i])$ be the sharings on the i-th multiplication gate for $i \in [1, N]$. All verifiers and $\mathcal{P}$ execute as follows:

1. The verifiers call the coin-tossing functionality $\mathcal{F}_{\mathsf{coin}}$ to generate a random element $\chi \in \mathbb{K}$, and then set the following inner-product tuple:
$$[\boldsymbol{x}] := \Big([x_1], \ldots, \chi^{N-1} \cdot [x_N]\Big)\,,\ [\boldsymbol{y}] := ([y_1], \ldots, [y_N])\,,\ [z] := \sum_{i \in [1,N]} \chi^{i-1} \cdot [z_i].$$
2. The verifiers and $\mathcal{P}$ call functionality $\mathcal{F}_{\mathsf{verifyprod}}$ on $([\boldsymbol{x}], [\boldsymbol{y}], [z])$ to check that $z = \boldsymbol{x} \odot \boldsymbol{y}$. If the verifiers receive abort from $\mathcal{F}_{\mathsf{verifyprod}}$, then they abort.

Verification of circuit output: Let $[\eta]$ be the input sharing associated with the single circuit-output gate. All verifiers execute $\eta \leftarrow \mathsf{Open}([\eta])$, and abort if outputting abort in the Open procedure. If $\eta = 0$, then the verifiers output true, otherwise they output false.

Fig. 3: **Information-theoretic NIMVZK in the $(\mathcal{F}_{\mathsf{coin}}, \mathcal{F}_{\mathsf{verifyprod}})$-hybrid model.**

or multiplication gate, the prover directly shares the output value to all verifiers. The verifiers can locally compute the shares on the output wires of addition gates. Then, all verifiers check the correctness of all multiplication gates by transforming multiplication triples into an inner-product tuple and then calling functionality $\mathcal{F}_{\mathsf{verifyprod}}$. In parallel, the verifiers also check correctness of the single circuit-output gate via running the Open procedure.

In Figure 2, we give the precise definition of functionality $\mathcal{F}_{\mathsf{verifyprod}}$. In particular, if the prover is honest, the adversary can only obtain the shares of corrupted verifiers from this functionality, which does not reveal any information on the secrets. In other words, this functionality naturally captures zero-knowledge. If the prover is corrupted, this functionality reveals all secrets to the adversary, as the secrets have been known anyway by the adversary. We can view deciding the correctness of an inner-product tuple as a statement which is shared among n verifiers. Functionality $\mathcal{F}_{\mathsf{verifyprod}}$ guarantees that the malicious prover cannot make any honest verifier accept a false statement, and thus captures soundness. In the full version [52], we present an efficient protocol to securely realize $\mathcal{F}_{\mathsf{verifyprod}}$, where the communication and round complexities are $O((n+\tau)\log_\tau |C|)$ field elements per verifier and $\log_\tau |C| + 3$ rounds between verifiers respectively, where $\tau \geq 2$ is a parameter.

Theorem 1. *Protocol $\Pi^{\mathsf{it}}_{\mathsf{nimvzk}}$ shown in Figure 3 securely realizes functionality $\mathcal{F}_{\mathsf{mvzk}}$ with information-theoretic security and soundness error $\frac{N-1}{|\mathbb{K}|}$ in the $(\mathcal{F}_{\mathsf{coin}}, \mathcal{F}_{\mathsf{verifyprod}})$-hybrid model in the presence of a malicious adversary corrupting up to a prover and t verifiers.*

The proof of this theorem can be found in the full version [52].

5 Strong NIMVZK Proof in the Honest-Majority Setting

In this section, we present a strong NIMVZK proof based on the Fiat-Shamir transform, where a minority of verifiers are allowed to be corrupted and collude with the prover. Our strong NIMVZK protocol adopts a non-interactive commitment based on random oracle to non-interactively transform the verification of multiplication triples into the verification of an inner-product tuple. This protocol still works in the $\mathcal{F}_{\mathsf{verifyprod}}$-hybrid model, where functionality $\mathcal{F}_{\mathsf{verifyprod}}$ can now be non-interactively realized using the Fiat-Shamir transform.

In Figure 4, we describe the strong NIMVZK protocol $\Pi^{\mathsf{fs}}_{\mathsf{snimvzk}}$ in the $\mathcal{F}_{\mathsf{verifyprod}}$-hybrid model, where the shares are computed over a field $\mathbb{F}$ and the verification of multiplication gates is performed over an extension field $\mathbb{K}$ with $|\mathbb{K}| \geq 2^\lambda$. The strong NIMVZK protocol is the same as the protocol shown in Figure 3, except for the verification of multiplication gates. In the strong NIMVZK protocol, the verification of multiplication gates is executed non-interactively using a non-interactive commitment based on a random oracle H_1, where a commitment *com* on a message x is defined as $\mathsf{H}_1(x, r)$ for a randomness $r \in \{0,1\}^\lambda$. However, we do *not* require that the commitment is binding. Instead, we only need the commitment to be hard to find a pair (x', r') such that $\mathsf{H}_1(x', r') = \mathsf{H}_1(x, r)$ and $x' \neq x$, after $\mathsf{H}_1(x, r)$ has been defined. This has been explained in Section 3.2 (see the proof of Theorem 2 for details). The random challenge $\chi \in \mathbb{K}$ is now generated using another random oracle H_2 and the public commitments, instead of calling $\mathcal{F}_{\mathsf{coin}}$. In this case, the prover can compute the secrets $(\boldsymbol{x}, \boldsymbol{y}, z)$ underlying the inner-product tuple using the public coefficient χ and the secret wire values. At first glance, the secrets $(\boldsymbol{x}, \boldsymbol{y}, z)$ seem to be useless for the protocol execution of $\Pi^{\mathsf{fs}}_{\mathsf{snimvzk}}$. Nevertheless, the prover can use $(\boldsymbol{x}, \boldsymbol{y}, z)$ to compute all the secrets involved in the protocol that securely realizes functionality $\mathcal{F}_{\mathsf{verifyprod}}$. In this case, we can securely compute $\mathcal{F}_{\mathsf{verifyprod}}$ in a strongly non-interactive way by making the prover distribute the shares of all secrets non-interactively and all verifiers interact only one round for Open.

Protocol $\Pi^{\mathsf{fs}}_{\mathsf{snimvzk}}$

Inputs: A prover $\mathcal{P}$ holds a witness $\boldsymbol{w} \in \mathbb{F}^m$. $\mathcal{P}$ and the verifiers $\mathcal{V}_1, \ldots, \mathcal{V}_n$ hold an arithmetic circuit C over a field $\mathbb{F}$ with $|\mathbb{F}| > n$ such that $C(\boldsymbol{w}) = 0$. Let N denote the number of multiplication gates in the circuit. Let $\mathsf{H}_1 : \{0,1\}^* \to \{0,1\}^\lambda$ and $\mathsf{H}_2 : \{0,1\}^* \to \mathbb{K}$ be two random oracles.

Circuit evaluation: $\mathcal{P}$ and all verifiers evaluate the circuit in the same way as described in Figure 3.

Verification of multiplication gates: For all m circuit-input wires, the verifiers $\mathcal{V}_1, \ldots, \mathcal{V}_n$ hold the shares of $[w_1], \ldots, [w_m]$, and $\mathcal{P}$ has the witness $\boldsymbol{w} = (w_1, \ldots, w_m)$. For all N multiplication gates, $\mathcal{P}$ and the verifiers respectively hold the secrets and the shares of multiplication triples $([x_1], [y_1], [z_1]), \ldots, ([x_N], [y_N], [z_N])$. For $j \in [1, m]$, let $w_j^1, \ldots, w_j^n$ be the shares of $[w_j]$ held by all verifiers, which are also known by $\mathcal{P}$. For $j \in [1, N]$, let $z_j^1, \ldots, z_j^n$ be the shares of $[z_j]$, which are also obtained by $\mathcal{P}$. All verifiers $\mathcal{V}_1, \ldots, \mathcal{V}_n$ and $\mathcal{P}$ execute as follows:

1. For each $i \in [1, n]$, $\mathcal{P}$ samples $r_i \leftarrow \{0,1\}^\lambda$ and computes
$$com_i := \mathsf{H}_1(w_1^i, \ldots, w_m^i, z_1^i, \ldots, z_N^i, r_i).$$
Then, $\mathcal{P}$ broadcasts $(com_1, \ldots, com_n)$ to all verifiers, and also sends r_i to every verifier $\mathcal{V}_i$ over a private channel.
2. Every verifier $\mathcal{V}_i$ checks that $com_i = \mathsf{H}_1(w_1^i, \ldots, w_m^i, z_1^i, \ldots, z_N^i, r_i)$, and aborts if the check fails.
3. $\mathcal{P}$ and all verifiers compute $\chi := \mathsf{H}_2(com_1, \ldots, com_n) \in \mathbb{K}$.
4. $\mathcal{P}$ and all verifiers respectively compute the secrets and the shares of the following inner-product tuple:
$$[\boldsymbol{x}] := ([x_1], \ldots, \chi^{N-1} \cdot [x_N]),\ [\boldsymbol{y}] := ([y_1], \ldots, [y_N]),\ [z] := \textstyle\sum_{i \in [1,N]} \chi^{i-1} \cdot [z_i].$$
5. The verifiers and $\mathcal{P}$ call functionality $\mathcal{F}_{\mathsf{verifyprod}}$ on $([\boldsymbol{x}], [\boldsymbol{y}], [z])$ to check that $z = \boldsymbol{x} \odot \boldsymbol{y}$. If the verifiers receive abort from $\mathcal{F}_{\mathsf{verifyprod}}$, then they abort.

Verification of circuit output: All verifiers check the correctness of a single circuit-output gate in the same way as shown in Figure 3.

Fig. 4: **Strong non-interactive MVZK protocol in the $\mathcal{F}_{\mathsf{verifyprod}}$-hybrid model and random oracle model.**

Theorem 2. *Let H_1 and H_2 be two random oracles. Protocol $\Pi^{\mathsf{fs}}_{\mathsf{snimvzk}}$ shown in Figure 4 securely realizes functionality $\mathcal{F}_{\mathsf{mvzk}}$ with soundness error at most $\frac{Q_1 n + (Q_2+1)N}{2^\lambda}$ in the $\mathcal{F}_{\mathsf{verifyprod}}$-hybrid model in the presence of a malicious adversary corrupting up to a prover and t verifiers, where Q_1 and Q_2 are the number of queries to random oracles H_1 and H_2 respectively.*

The proof of the above theorem is given in the full version [52].

Optimizations. For Shamir secret sharing , $\mathcal{P}$ can send a random $\mathsf{seed}_i \in \{0,1\}^\lambda$ to $\mathcal{V}_i$ for each $i \in [1,t]$, who computes all its shares with seed_i and a pseudo-random generator (PRG). This reduces the communication by a half. Furthermore, for each $i \in [1,t]$, $\mathcal{P}$ can send $com_i = \mathsf{H}_1(\mathsf{seed}_i, r_i)$ to $\mathcal{V}_i$, who checks the correctness of com_i using seed_i and r_i. This will reduce the computational cost of generating and verifying t commitments. Using the optimization, the communication among verifiers is *asymmetry*. In particular, among all verifiers, t verifiers $\mathcal{V}_1, \ldots, \mathcal{V}_t$ only has a *sublinear* communication complexity. That is, each verifier only receives $O(n + \log|C|)$ field elements from the prover, and only needs to send $O(n)$ field elements to other verifiers. It makes our strong NIMVZK protocol particularly suitable for the applications where $\mathcal{V}_1, \ldots, \mathcal{V}_t$ are lower-resource mobile devices and the other verifiers are powerful servers.

Strong NIMVZK proof for inner-product tuples. Boneh et al. [14] introduced a powerful tool, called distributed zero-knowledge (DZK) proof (a.k.a., ZK proof on a distributed or secret-shared statement), to prove the inner-product statements (and other useful statements). We can use their DZK proof with logarithmic communication to securely realize functionality $\mathcal{F}_{\mathsf{verifyprod}}$ shown in Figure 2. When applying the Fiat-Shamir transform [16] into their DZK proof, the prover non-interactively sends a proof to all verifiers, and the verifiers execute one-round communication to verify correctness of an inner-product tuple. Note that the proof on the inner-product statement can be sent in parallel with our proof on circuit satisfiability shown in Figure 4. Therefore, using the DZK proof to instantiate $\mathcal{F}_{\mathsf{verifyprod}}$, our MVZK protocol is strongly non-interactive. While Boneh et al. [14] originally instantiated the DZK proof with replicated secret sharing, Boyle et al. [16] shown that their DZK proof also works for verifiable Shamir secret sharing meaning that a consistency check is needed to guarantee either all verifiers hold a consistent sharing of the secret or honest verifiers abort.

We can simplify the technique by Boneh et al. [14] by avoiding the use of verifiable secret sharing, and slightly optimize the communication from $4.5\log|C| + 5n$ field elements to $3\log|C| + 3n$ field elements. We can also improve the hash computation cost for Fiat-Shamir. The improved approach has been described in Section 3 by considering Shamir secret sharing as a special case of packed secret sharing. The detailed protocol to strongly non-interactively realize $\mathcal{F}_{\mathsf{verifyprod}}$ can be directly obtained by simplifying the PSS-based protocol $\Pi^{\mathsf{pss}}_{\mathsf{verifyprod}}$ shown in Figure 8 of Section 6.2 via setting the number of packed secrets $k = 1$.

6 Strong NIMVZK Proof with Lower Communication

Based on packed secret sharing (PSS), we present a strong NIMVZK proof with communication complexity $O(|C|/n)$ per verifier, when the threshold of corrupted verifiers $t < n(1/2-\epsilon)$ for any $0 < \epsilon < 1/2$. Our strong NIMVZK protocol is highly efficient for proving satisfiability of a *single generic* circuit. In the ZK setting, we use PSS optimally. In particular, we eliminate the constraints in the state-of-the-art PSS-based MPC protocol [39] including: 1) evaluating a circuit layer-by-layer, 2) interactively permuting the secrets in a single packed sharing,

Functionality $\mathcal{F}_{\mathsf{verifyprod}}^{\mathsf{pss}}$

The packed inner-product tuple over a field $\mathbb{K}$ is denoted by $([\boldsymbol{x}_1], \ldots, [\boldsymbol{x}_\ell])$, $([\boldsymbol{y}_1], \ldots, [\boldsymbol{y}_\ell])$ and $[\boldsymbol{z}]$. This functionality runs with a prover $\mathcal{P}$ and n verifiers $\mathcal{V}_1, \ldots, \mathcal{V}_n$, and operates as follows:

1. Upon receiving the shares of $\{[\boldsymbol{x}_i], [\boldsymbol{y}_i]\}_{i \in [1,\ell]}$ and $[\boldsymbol{z}]$ from all honest verifiers, execute the following:
   - Reconstruct the secrets $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_\ell)$, $(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_\ell)$ and $\boldsymbol{z}$ from the shares of honest verifiers in $\mathcal{H}_H$.
   - Compute the shares of corrupted verifiers on $([\boldsymbol{x}_1], \ldots, [\boldsymbol{x}_\ell])$, $([\boldsymbol{y}_1], \ldots, [\boldsymbol{y}_\ell])$ and $[\boldsymbol{z}]$, and then send these shares to the adversary.
   - If $\mathcal{P}$ is corrupted, send the shares of $\big(\{[\boldsymbol{x}_i], [\boldsymbol{y}_i]\}_{i \in [1,\ell]}, [\boldsymbol{z}]\big)$ held by honest verifiers in $\mathcal{H}$ and the secrets $\big(\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i \in [1,\ell]}, \boldsymbol{z}\big)$ to the adversary.
2. If $\boldsymbol{z} \neq \sum_{h \in [1,\ell]} \boldsymbol{x}_h * \boldsymbol{y}_h$ where $*$ denotes the component-wise product, then set $b := \mathsf{abort}$, otherwise set $b := \mathsf{accept}$. Send b to the adversary. For $i \in \mathcal{H}$, wait for an input from the adversary, and do the following:
   - If it is $\mathsf{continue}_i$, send b to $\mathcal{V}_i$.
   - If it is abort_i, send abort to $\mathcal{V}_i$.

Fig. 5: **ZK verification functionality for packed inner-product tuples.**

3) interactively collecting the secrets from different packed sharings and 4) splitting an output wire into multiple output wires, where all these constraints will make the rounds and communication cost significantly larger than our protocol.

Firstly, we discuss how to transform a general circuit C into another circuit C' with the same output and $|C'| = |C| + O(k)$, such that 1) the number of circuit-input wires, addition gates and multiplication gates is the multiple of k; 2) there are at least k circuit-output wires; 3) the gates with the same type are divided into groups of k. This is done by adding "dummy" wires and gates, and is described in the full version [52]. Then, we present the detailed strong NIMVZK protocol in the $\mathcal{F}_{\mathsf{verifyprod}}^{\mathsf{pss}}$-hybrid model, where $\mathcal{F}_{\mathsf{verifyprod}}^{\mathsf{pss}}$ verifies the correctness of a packed inner-product tuple. Next, we present a strong non-interactive MVZK protocol to securely realize functionality $\mathcal{F}_{\mathsf{verifyprod}}^{\mathsf{pss}}$.

6.1 Strong NIMVZK based on Packed Secret Sharing

Before showing the detailed strong NIMVZK protocol, we give the definition of functionality $\mathcal{F}_{\mathsf{verifyprod}}^{\mathsf{pss}}$.

Functionality for verifying packed inner-product tuples. Let $\mathcal{H}_H \subset \mathcal{H}$ be a fixed $(d+1)$-sized subset of honest verifiers and $\mathcal{H}_C = \mathcal{H} \backslash \mathcal{H}_H$, where recall that $\mathcal{H}$ is the set of all $d+k$ honest verifiers and d is the degree of polynomials for PSS. For a degree-d packed sharing $[\boldsymbol{x}]$, we use $[\boldsymbol{x}]_{\mathcal{H}}$ to denote the whole sharing that is reconstructed from the shares of honest verifiers in $\mathcal{H}_H$. Given the shares of honest verifiers in $\mathcal{H}$ as input, we can reconstruct the *whole* sharing $[\boldsymbol{x}]$ as follows:

1. Use the $d+1$ shares of honest verifiers in $\mathcal{H}_H$ to reconstruct the whole sharing $[\boldsymbol{x}]_{\mathcal{H}}$. Define the shares of corrupted verifiers on $[\boldsymbol{x}]$ as that on $[\boldsymbol{x}]_{\mathcal{H}}$. Following prior MPC work [39], we always assume that the corrupted verifiers in $\mathcal{C}$ hold the correct shares that they should hold, while they may use incorrect shares during the protocol execution, where $\mathcal{C}$ is the set of corrupted verifiers.
2. Define the secrets of $[\boldsymbol{x}]$ to be that of $[\boldsymbol{x}]_{\mathcal{H}}$.
3. Define the shares of $[\boldsymbol{x}]$ held by honest verifiers in $\mathcal{H}$ as the shares input by the verifiers directly.

The zero-knowledge verification functionality for packed inner-product tuples is shown in Figure 5. This functionality takes as input a packed inner-product tuple and then checks correctness of the tuple, where each sharing packs k secrets. This functionality sends the shares of corrupted verifiers for each packed sharing to the adversary, where the shares are computed by the above approach based on the shares of honest verifiers in $\mathcal{H}_H$. If the prover is corrupted, this functionality also sends the shares of all honest verifiers and the secrets in all packed sharings to the adversary, as these shares and secrets have been known by the adversary.

PSS-based strong NIMVZK protocol from the Fiat-Shamir transform. Our PSS-based strong non-interactive MVZK protocol in the $\mathcal{F}^{\mathsf{pss}}_{\mathsf{verifyprod}}$-hybrid model is described in Figures 6 and 7, where the circuit is defined over a field $\mathbb{F}$ and the verification is performed over an extension field $\mathbb{K}$ with $|\mathbb{K}| \geq 2^{\lambda}$.

The prover and all verifiers first transform the circuit C into an *equivalent* circuit C', which satisfies the requirements of packed secret sharings. For an input vector $\boldsymbol{w} \in \mathbb{F}^k$, if the j-th secret of $\boldsymbol{w}$ for $j \in [1, k]$ corresponds to a dummy circuit-input wire, the secret is set as 0. If $\boldsymbol{w}$ corresponds to k dummy circuit-input wires, then $\boldsymbol{w} = 0^k$ and $[\boldsymbol{w}]$ can be *locally* generated by all verifiers without any communication (as shown in the full version [52]).

Using the non-interactive commitment based on a random oracle, we adopt a similar approach as described in the previous section to transform the check of multiplication tuples into the check of a packed inner-product tuple. Then, by calling functionality $\mathcal{F}^{\mathsf{pss}}_{\mathsf{verifyprod}}$, the verifiers can check correctness of the packed inner-product tuple. Note that the prover can compute the secrets of the packed inner-product tuple, which will be useful for designing a strongly non-interactive protocol to securely realize $\mathcal{F}^{\mathsf{pss}}_{\mathsf{verifyprod}}$ as shown in Section 6.2.

During the protocol execution, we need to check the consistency of some secrets stored in two different packed sharings. We perform the consistency check using the random-linear-combination approach based on the Fiat-Shamir transform, which is inspired by the recent checking approach by Goyal et al. [39] for information-theoretic MPC. In particular, the prover will generate a packed input sharing $[\boldsymbol{y}]$ on k multiplication gates, addition gates or circuit-output gates, such that the j-th secret y_j of $[\boldsymbol{y}]$ comes from the i-th secret x_i of a packed output sharing $[\boldsymbol{x}]$ of k circuit-input gates, multiplication gates or addition gates. We need to check that $x_i = y_j$ to guarantee the consistency of y_j. This corresponds to the wire which carries the value $x_i = y_j$ in the circuit. We refer to a tuple $([\boldsymbol{x}], [\boldsymbol{y}], i, j)$ as a *wire tuple* following prior work [39]. We perform the

Protocol $\Pi^{\mathsf{pss}}_{\mathsf{snimvzk}}$

Inputs: A prover $\mathcal{P}$ holds a witness $\bar{\boldsymbol{w}}$. $\mathcal{P}$ and n verifiers $\mathcal{V}_1, \ldots, \mathcal{V}_n$ hold a circuit C over a field $\mathbb{F}$. Let k denote the number of secrets that are packed in a single sharing. Let m be the number of packed sharings on circuit-input wires, where each packs k circuit-input gates. Let $M = \lceil |C|/k \rceil$ denote the number of multiplication tuples where each packs k multiplication gates. Let $N = O(|C|/k)$ represent the number of wire tuples in the form of $([\boldsymbol{x}], [\boldsymbol{y}], i, j)$ with $x_i = y_j$. Let $\mathsf{H}_1 : \{0,1\}^* \to \{0,1\}^\lambda$ and $\mathsf{H}_2 : \{0,1\}^* \to \mathbb{K}$ be two random oracles.

Preprocess circuit: All parties run the $\mathsf{PrepCircuit}$ procedure shown in the full version [52] to preprocess the circuit C, and obtain an equivalent circuit C' such that $C'(\bar{\boldsymbol{w}}) = C(\bar{\boldsymbol{w}})$ for any input $\bar{\boldsymbol{w}}$.

Circuit evaluation: In a predetermined topological order, $\mathcal{P}$ and all verifiers evaluate the circuit C' as follows:

- Let $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_m \in \mathbb{F}^k$ be the secret vectors where each associates with k circuit-input wires. For each group of k circuit-input wires with an input vector $\boldsymbol{w} \in \{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_m\}$, $\mathcal{P}$ runs $[\boldsymbol{w}] \leftarrow \mathsf{Share}(\boldsymbol{w})$ to distribute the shares to all verifiers.
- For each group of k addition gates with input sharings $[\boldsymbol{x}]$ and $[\boldsymbol{y}]$, all verifiers *locally* compute $[\boldsymbol{z}] := [\boldsymbol{x}] + [\boldsymbol{y}]$, and $\mathcal{P}$ computes $\boldsymbol{z} := \boldsymbol{x} + \boldsymbol{y} \in \mathbb{F}^k$.
- For each group of k multiplication gates with input sharings $[\boldsymbol{x}]$ and $[\boldsymbol{y}]$, $\mathcal{P}$ computes $\boldsymbol{z} := \boldsymbol{x} * \boldsymbol{y} \in \mathbb{F}^k$ where $*$ denotes the component-wise product, and executes $[\boldsymbol{z}] \leftarrow \mathsf{Share}(\boldsymbol{z})$ which distributes the shares to all verifiers.
- For each group of k input wires of multiplication, addition, or circuit-output gates, such that the corresponding input vector $\boldsymbol{y} \in \mathbb{F}^k$ has not been stored in any single packed input sharing, $\mathcal{P}$ runs $[\boldsymbol{y}] \leftarrow \mathsf{Share}(\boldsymbol{y})$, which distributes the shares to all verifiers.

Locally prepare packed sharings. All verifiers locally do the following:

- For each input sharing $[\boldsymbol{y}]$ on a group of k multiplication, addition, or circuit-output gates, for each position $j \in [1, k]$, suppose that y_j comes from the i-th secret of $[\boldsymbol{x}]$ that is an output sharing on a group of k circuit-input, multiplication, or addition gates. If $[\boldsymbol{y}] \neq [\boldsymbol{x}]$, then set a wire tuple as $([\boldsymbol{x}], [\boldsymbol{y}], i, j)$.
- Denote these wire tuples by $([\boldsymbol{x}_1], [\boldsymbol{y}_1], i_1, j_1), \ldots, ([\boldsymbol{x}_N], [\boldsymbol{y}_N], i_N, j_N)$.
- Remove the repetitive packed sharings in $[\boldsymbol{y}_1], \ldots, [\boldsymbol{y}_N]$, and then denote the resulting sharings as $[\boldsymbol{y}'_1], \ldots, [\boldsymbol{y}'_\ell]$, where ℓ denotes the number of different packed sharings in $[\boldsymbol{y}_1], \ldots, [\boldsymbol{y}_N]$.

All verifiers hold the shares of the following packed sharings:

- The shares of $[\boldsymbol{w}_1], \ldots, [\boldsymbol{w}_m]$ on all circuit-input wires, which are denoted by $(\hat{w}_i^1, \ldots, \hat{w}_i^n)$ for $i \in [1, m]$.
- Let $([\boldsymbol{x}_i], [\boldsymbol{y}_i], [\boldsymbol{z}_i])$ be the i-th multiplication tuple packing the secrets of k multiplication gates for $i \in [1, M]$. The shares of $\{([\boldsymbol{x}_i], [\boldsymbol{y}_i], [\boldsymbol{z}_i])\}_{i \in [1,M]}$, where $\hat{z}_i^1, \ldots, \hat{z}_i^n$ denote the shares of $[\boldsymbol{z}_i]$ for $i \in [1, M]$.
- The shares of $\{[\boldsymbol{y}'_i]\}_{i \in [1,\ell]}$, which are denoted by $\hat{y}_i^1, \ldots, \hat{y}_i^n$ for $i \in [1, \ell]$.

$\mathcal{P}$ holds the whole sharings $\{[\boldsymbol{w}_i]\}_{i \in [1,m]}$, $\{([\boldsymbol{x}_i], [\boldsymbol{y}_i], [\boldsymbol{z}_i])\}_{i \in [1,M]}$ and $\{[\boldsymbol{y}'_i]\}_{i \in [1,\ell]}$.

Fig. 6: **PSS-based strong NIMVZK in the $\mathcal{F}^{\mathsf{pss}}_{\mathsf{verifyprod}}$-hybrid model and random oracle model.**

Protocol $\Pi^{\mathsf{pss}}_{\mathsf{snimvzk}}$, continued

Procedure for Fiat-Shamir: All verifiers $\mathcal{V}_1, \ldots, \mathcal{V}_n$ and $\mathcal{P}$ execute as follows:

1. For $i \in [1, n]$, $\mathcal{P}$ samples $r_i \leftarrow \{0,1\}^\lambda$ and computes
$$com_i := \mathsf{H}_1(\hat{w}^i_1, \ldots, \hat{w}^i_m, \hat{z}^i_1, \ldots, \hat{z}^i_M, \hat{y}^i_1, \ldots, \hat{y}^i_\ell, r_i).$$
Then, $\mathcal{P}$ broadcasts $(com_1, \ldots, com_n)$ to all verifiers, and sends r_i to every verifier $\mathcal{V}_i$ over a private channel.
2. Each verifier $\mathcal{V}_i$ checks that $com_i = \mathsf{H}_1(\hat{w}^i_1, \ldots, \hat{w}^i_m, \hat{z}^i_1, \ldots, \hat{z}^i_M, \hat{y}^i_1, \ldots, \hat{y}^i_\ell, r_i)$, and aborts if the check fails.
3. $\mathcal{P}$ and all verifiers compute $\chi := \mathsf{H}_2(com_1, \ldots, com_n) \in \mathbb{K}$.

Verification of multiplication tuples: $\mathcal{P}$ and all verifiers execute as follows:

1. $\mathcal{P}$ and all verifiers respectively compute the secrets and the shares of $[\tilde{\boldsymbol{x}}_i] = \chi^{i-1} \cdot [\boldsymbol{x}_i]$, $[\tilde{\boldsymbol{y}}_i] = [\boldsymbol{y}_i]$ for $i \in [1, M]$ and $[\tilde{\boldsymbol{z}}] := \sum_{i \in [1,M]} \chi^{i-1} \cdot [\boldsymbol{z}_i]$.
2. The verifiers and $\mathcal{P}$ call functionality $\mathcal{F}^{\mathsf{pss}}_{\mathsf{verifyprod}}$ on packed inner-product tuple $(([\tilde{\boldsymbol{x}}_1], \ldots, [\tilde{\boldsymbol{x}}_M]), ([\tilde{\boldsymbol{y}}_1], \ldots, [\tilde{\boldsymbol{y}}_M]), [\tilde{\boldsymbol{z}}])$ to check that $\tilde{\boldsymbol{z}} = \sum_{h \in [1,M]} \tilde{\boldsymbol{x}}_h * \tilde{\boldsymbol{y}}_h$. If the verifiers receive abort from $\mathcal{F}^{\mathsf{pss}}_{\mathsf{verifyprod}}$, then they abort.

Verification of consistency of wire tuples: For all $i, j \in [1, k]$, $\mathcal{P}$ and all verifiers initiate an empty list $L(i, j)$. Then, from $h = 1$ to N, they insert $([\boldsymbol{x}_h], [\boldsymbol{y}_h], i_h, j_h)$ into the list $L(i_h, j_h)$. For each $i, j \in [1, k]$, $\mathcal{P}$ and all verifiers check the consistency of the wire tuples in $L(i, j)$ as follows:

1. Let N' be the size of $L(i, j)$. Let $([\boldsymbol{a}_1], [\boldsymbol{b}_1], i, j), \ldots, ([\boldsymbol{a}_{N'}], [\boldsymbol{b}_{N'}], i, j)$ denote the wire tuples in $L(i, j)$.
2. $\mathcal{P}$ samples $\boldsymbol{a}_0, \boldsymbol{b}_0 \leftarrow \mathbb{K}^k$ with $a_{0,i} = b_{0,j}$, and also picks $\boldsymbol{r}, \boldsymbol{r}' \leftarrow \mathbb{K}^k$. Then $\mathcal{P}$ and all verifiers execute the following:
   (a) $\mathcal{P}$ runs $[\boldsymbol{r}] \leftarrow \mathsf{Share}(\boldsymbol{r})$ and $[\boldsymbol{r}'] \leftarrow \mathsf{Share}(\boldsymbol{r}')$, which distributes the shares to all verifiers.
   (b) $\mathcal{P}$ computes $\boldsymbol{u} := \boldsymbol{a}_0 + \boldsymbol{r}$ and $\boldsymbol{u}' := \boldsymbol{b}_0 + \boldsymbol{r}'$, and then broadcasts $(\boldsymbol{u}, \boldsymbol{u}')$ to all verifiers.
   (c) The verifiers compute $[\boldsymbol{a}_0] := \boldsymbol{u} - [\boldsymbol{r}]$ and $[\boldsymbol{b}_0] := \boldsymbol{u}' - [\boldsymbol{r}']$.
3. All verifiers compute $\alpha := \mathsf{H}_2(\chi, \boldsymbol{u}, \boldsymbol{u}', i, j) \in \mathbb{K}$.
4. All verifiers compute $[\boldsymbol{a}] := \sum_{h=0}^{N'} \alpha^h \cdot [\boldsymbol{a}_h]$ and $[\boldsymbol{b}] := \sum_{h=0}^{N'} \alpha^h \cdot [\boldsymbol{b}_h]$.
5. All verifiers run $\boldsymbol{a} \leftarrow \mathsf{Open}([\boldsymbol{a}])$ and $\boldsymbol{b} \leftarrow \mathsf{Open}([\boldsymbol{b}])$. If abort is output for the Open procedure, the verifiers abort. Otherwise, they check that $a_i = b_j$, and abort if the check fails.

Verification of circuit output: Let $[\boldsymbol{z}]$ be the sharing on the group of k circuit-output wires including the single actual circuit-output wire. All verifiers execute $\boldsymbol{z} \leftarrow \mathsf{Open}([\boldsymbol{z}])$, and abort if receiving abort from the Open procedure. If $\boldsymbol{z} = 0^k$, then the verifiers output true, otherwise they output false.

Fig. 7: **PSS-based strong NIMVZK in the $\mathcal{F}^{\mathsf{pss}}_{\mathsf{verifyprod}}$-hybrid model and random oracle model, continued.**

consistency check of wire tuples in the *total* communication complexity $O(nk^2)$ elements between the prover and verifiers and $O(n^2k^2)$ elements among verifiers.

Theorem 3. *Let* H_1 *and* H_2 *be two random oracles. Protocol* $\Pi^{\mathsf{pss}}_{\mathsf{snimvzk}}$ *shown in Figures 6 and 7 securely realizes functionality* $\mathcal{F}_{\mathsf{mvzk}}$ *with soundness error at most* $\frac{Q_1 n + (Q_2+1)(M+N)}{2^\lambda}$ *in the* $\mathcal{F}^{\mathsf{pss}}_{\mathsf{verifyprod}}$*-hybrid model in the presence of a malicious adversary corrupting up to a prover and* $t = d - k + 1$ *verifiers, where degree-d packed sharings are used in protocol* $\Pi^{\mathsf{pss}}_{\mathsf{snimvzk}}$*, each sharing packs* k *secrets, and* Q_1 *and* Q_2 *are the number of queries to* H_1 *and* H_2 *respectively.*

The proof of Theorem 3 can be found in the full version [52].

Protocol $\Pi^{\mathsf{pss}}_{\mathsf{snimvzk}}$ shown in Figures 6 and 7 are *streamable*, i.e., the circuit can be proved on-the-fly. The prover $\mathcal{P}$ can prove a batch of addition and multiplication gates each time, and stores the secrets that will be used as the input wire values in the next batches of gates. In Section 3.3, we give an approach overview on how to stream our protocol $\Pi^{\mathsf{pss}}_{\mathsf{snimvzk}}$ with the *same* round among verifiers. In the full version [52], we provide more details.

6.2 Strong NIMVZK Proof for Packed Inner-Product Tuples

Below, we present a strongly non-interactive MVZK protocol with *logarithmic* communication complexity to verify packed inner-product tuples, which is inspired by the technique by Goyal et al. [39] for MPC that is in turn built on the techniques [14,40,41]. While the MPC protocol [39] requires logarithmic rounds to check correctness of packed inner-product tuples, our strong NIMVZK protocol needs only one round between verifiers. Furthermore, our protocol reduces the communication overhead for verification by making the prover generate the random sharings and messages associated with secrets, compared to the verification of packed inner-product tuples directly using the MPC protocol [39].

Our PSS-based strong NIMVZK protocol $\Pi^{\mathsf{pss}}_{\mathsf{verifyprod}}$ for verifying a packed inner-product tuple is described in Figure 8. This protocol invokes two sub-protocols $\Pi^{\mathsf{pss}}_{\mathsf{inner\text{-}prod}}$ and $\Pi^{\mathsf{pss}}_{\mathsf{compress}}$ that are described in Figures 9 and 10 respectively, where $\Pi^{\mathsf{pss}}_{\mathsf{inner\text{-}prod}}$ is used to generate the inner product of two vectors and $\Pi^{\mathsf{pss}}_{\mathsf{compress}}$ is used to compress two packed inner-product tuples into a single tuple. In the dimension-reduction and randomization phases of this protocol, we adapt the approach by Baum et al. [8] used in the DVZK setting to non-interactively generate a challenge α used in sub-protocol $\Pi^{\mathsf{pss}}_{\mathsf{compress}}$ based on the Fiat-Shamir transform. Based on the *round-by-round soundness* [20,8], we can prove that the soundness error of our protocol is negligible (see Theorem 4). In the dimension-reduction phase of $\Pi^{\mathsf{pss}}_{\mathsf{verifyprod}}$, we always assume that the dimension m of the packed inner-product tuple is the multiple of 2 for each iteration. If not, we can pad the dummy zero sharing $[\mathbf{0}]$ into the packed inner-product tuple to satisfy the requirement, where $[\mathbf{0}]$ can be locally computed by all verifiers.

In the protocol $\Pi^{\mathsf{pss}}_{\mathsf{verifyprod}}$ shown in Figure 8, we assume that $\mathcal{P}$ and all verifiers input a public challenge χ, which is determined after the secrets packed in the input inner-product tuple $(([\boldsymbol{x}_1], \ldots, [\boldsymbol{x}_M]), ([\boldsymbol{y}_1], \ldots, [\boldsymbol{y}_M]), [\boldsymbol{z}])$ have been

Protocol $\Pi^{\mathsf{pss}}_{\mathsf{verifyprod}}$

Inputs: Prover $\mathcal{P}$ and all verifiers $\mathcal{V}_1, \ldots, \mathcal{V}_n$ respectively hold the secrets and shares of a packed inner-product tuple $(([\boldsymbol{x}_1], \ldots, [\boldsymbol{x}_M]), ([\boldsymbol{y}_1], \ldots, [\boldsymbol{y}_M]), [\boldsymbol{z}])$. $\mathcal{P}$ and all verifiers also hold a public value χ, which is determined after these secrets have been defined. The verifiers will check that $\boldsymbol{z} = \sum_{h \in [1,M]} \boldsymbol{x}_h * \boldsymbol{y}_h \in \mathbb{K}^k$. Let $\mathsf{H}_2 : \{0,1\}^* \to \mathbb{K}$ be a random oracle where $|\mathbb{K}| \geq 2^\lambda$.

- **Dimension-reduction:** Let m denote the dimension of the packed inner-product tuple in the current iteration, where m is initialized as M. Let γ be the public value in the current iteration, which is initialized as χ.
  While $m > 2$, $\mathcal{P}$ and all verifiers do the following:
  1. Let $\ell = m/2$. $\mathcal{P}$ and all verifiers define $([\boldsymbol{a}_{1,1}], \ldots, [\boldsymbol{a}_{1,\ell}]) := ([\boldsymbol{x}_1], \ldots, [\boldsymbol{x}_\ell])$ and $([\boldsymbol{a}_{2,1}], \ldots, [\boldsymbol{a}_{2,\ell}]) = ([\boldsymbol{x}_{\ell+1}], \ldots, [\boldsymbol{x}_m])$, where $\mathcal{P}$ holds all the secrets and the verifiers hold the shares. Similarly, they define $([\boldsymbol{b}_{1,1}], \ldots, [\boldsymbol{b}_{1,\ell}]) := ([\boldsymbol{y}_1], \ldots, [\boldsymbol{y}_\ell])$ and $([\boldsymbol{b}_{2,1}], \ldots, [\boldsymbol{b}_{2,\ell}]) = ([\boldsymbol{y}_{\ell+1}], \ldots, [\boldsymbol{y}_m])$.
  2. $\mathcal{P}$ and all verifiers execute the sub-protocol $\Pi^{\mathsf{pss}}_{\mathsf{inner\text{-}prod}}$ (shown in Figure 9) on $([\boldsymbol{a}_{1,1}], \ldots, [\boldsymbol{a}_{1,\ell}])$ and $([\boldsymbol{b}_{1,1}], \ldots, [\boldsymbol{b}_{1,\ell}])$ to compute the whole sharing $[\boldsymbol{c}_1] = [\sum_{h \in [1,\ell]} \boldsymbol{a}_{1,h} * \boldsymbol{b}_{1,h}]$, where the secrets and the shares are output to $\mathcal{P}$ and the verifiers respectively. $\mathcal{P}$ and all verifiers also obtain $\boldsymbol{u} \in \mathbb{K}^k$.
  3. $\mathcal{P}$ and all verifiers compute $[\boldsymbol{c}_2] := [\boldsymbol{z}] - [\boldsymbol{c}_1]$, where $\mathcal{P}$ obtains $\boldsymbol{c}_2$ and the verifiers get the shares of $[\boldsymbol{c}_2]$.
  4. $\mathcal{P}$ and all verifiers execute sub-protocol $\Pi^{\mathsf{pss}}_{\mathsf{compress}}$ on $([\boldsymbol{a}_{i,1}], \ldots, [\boldsymbol{a}_{i,\ell}])$, $([\boldsymbol{b}_{i,1}], \ldots, [\boldsymbol{b}_{i,\ell}])$ and $[\boldsymbol{c}_i]$ for $i \in [1,2]$ and $(\gamma, \boldsymbol{u})$, where $\Pi^{\mathsf{pss}}_{\mathsf{compress}}$ is described in Figure 10.
  5. $\mathcal{P}$ and all verifiers update γ as the public element α output by $\Pi^{\mathsf{pss}}_{\mathsf{compress}}$, and also set $m := m/2$. Then, $\mathcal{P}$ and the verifiers use the whole output sharings $(([\boldsymbol{a}_1], \ldots, [\boldsymbol{a}_\ell]), ([\boldsymbol{b}_1], \ldots, [\boldsymbol{b}_\ell]), [\boldsymbol{c}])$ from $\Pi^{\mathsf{pss}}_{\mathsf{compress}}$ to update $(([\boldsymbol{x}_1], \ldots, [\boldsymbol{x}_m]), ([\boldsymbol{y}_1], \ldots, [\boldsymbol{y}_m]), [\boldsymbol{z}])$.
- **Randomization:** Let $\gamma \in \mathbb{K}$ along with the inner-product tuple $(([\boldsymbol{x}_1], [\boldsymbol{x}_2]), ([\boldsymbol{y}_1], [\boldsymbol{y}_2]), [\boldsymbol{z}])$ be the final output from the previous phase. $\mathcal{P}$ and all verifiers execute the following procedure:
  1. $\mathcal{P}$ samples $\boldsymbol{x}_0, \boldsymbol{y}_0 \leftarrow \mathbb{K}^k$, and then runs $[\boldsymbol{x}_0] \leftarrow \mathsf{Share}(\boldsymbol{x}_0)$ and $[\boldsymbol{y}_0] \leftarrow \mathsf{Share}(\boldsymbol{y}_0)$, which distribute the shares to all verifiers.
  2. For $i \in [0,1]$, $\mathcal{P}$ and all verifiers execute the sub-protocol $\Pi^{\mathsf{pss}}_{\mathsf{inner\text{-}prod}}$ on $([\boldsymbol{x}_i], [\boldsymbol{y}_i])$ to compute the whole sharing $[\boldsymbol{z}_i] = [\boldsymbol{x}_i * \boldsymbol{y}_i]$. Additionally, $\mathcal{P}$ and the verifiers also obtain $\boldsymbol{u}_0, \boldsymbol{u}_1 \in \mathbb{K}^k$.
  3. $\mathcal{P}$ and all verifiers compute $[\boldsymbol{z}_2] := [\boldsymbol{z}] - [\boldsymbol{z}_1]$, where $\mathcal{P}$ obtains $\boldsymbol{z}_2$ and the verifiers get the shares of $[\boldsymbol{z}_2]$.
  4. $\mathcal{P}$ and all verifiers execute sub-protocol $\Pi^{\mathsf{pss}}_{\mathsf{compress}}$ on $\{([\boldsymbol{x}_i], [\boldsymbol{y}_i], [\boldsymbol{z}_i])\}_{i \in [0,2]}$ and $(\gamma, \boldsymbol{u}_0, \boldsymbol{u}_1)$. Then, all verifiers obtain the output $([\boldsymbol{a}], [\boldsymbol{b}], [\boldsymbol{c}])$.
  5. All verifiers run $\boldsymbol{v} \leftarrow \mathsf{Open}([\boldsymbol{v}])$ for each $\boldsymbol{v} \in \{\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}\}$. If abort is received during the Open procedure, then the verifiers abort. Then, the verifiers check that $\boldsymbol{c} = \boldsymbol{a} * \boldsymbol{b}$. If the check fails, the verifiers output abort. Otherwise, they output accept.

Fig. 8: **One-round ZK verification protocol for packed inner-product tuples.**

Protocol $\Pi^{\mathsf{pss}}_{\mathsf{inner\text{-}prod}}$

Inputs: Prover $\mathcal{P}$ and verifiers $\mathcal{V}_1, \ldots, \mathcal{V}_n$ respectively hold the secrets and the shares of packed sharings $([\boldsymbol{x}_1], \ldots, [\boldsymbol{x}_\ell])$ and $([\boldsymbol{y}_1], \ldots, [\boldsymbol{y}_\ell])$ over a field $\mathbb{K}$.

Protocol execution: $\mathcal{P}$ and all verifiers execute as follows:

1. $\mathcal{P}$ samples $\boldsymbol{r} \leftarrow \mathbb{K}^k$, and then runs $[\boldsymbol{r}] \leftarrow \mathsf{Share}(\boldsymbol{r})$ that distributes the shares to all verifiers.
2. $\mathcal{P}$ computes $\boldsymbol{z} := \sum_{h \in [1,\ell]} \boldsymbol{x}_h * \boldsymbol{y}_h \in \mathbb{K}^k$, and then broadcasts $\boldsymbol{u} := \boldsymbol{z} + \boldsymbol{r} \in \mathbb{K}^k$ to all verifiers.
3. The verifiers compute $[\boldsymbol{z}] := \boldsymbol{u} - [\boldsymbol{r}]$. Then, $\mathcal{P}$ and the verifiers respectively output the secrets and shares of $[\boldsymbol{z}]$, and also output $\boldsymbol{u} \in \mathbb{K}^k$.

Fig. 9: **Non-interactive inner-product protocol for packed sharings secure up to additive errors.**

Protocol $\Pi^{\mathsf{pss}}_{\mathsf{compress}}$

Inputs: Let m be the number of packed inner-product tuples, and ℓ be the dimension of each packed inner-product tuple. For $i \in [1, m]$, prover $\mathcal{P}$ and all verifiers $\mathcal{V}_1, \ldots, \mathcal{V}_n$ respectively hold the secrets and shares of packed inner-product tuple $(([\boldsymbol{x}_{i,1}], \ldots, [\boldsymbol{x}_{i,\ell}]), ([\boldsymbol{y}_{i,1}], \ldots, [\boldsymbol{y}_{i,\ell}]), [\boldsymbol{z}_i])$. $\mathcal{P}$ and all verifiers also input $\gamma \in \mathbb{K}$ and $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_d \in \mathbb{K}^k$. Let $\mathsf{H}_2 : \{0,1\}^* \to \mathbb{K}$ be a random oracle.

Protocol execution: $\mathcal{P}$ and all verifiers execute as follows:

1. For each $j \in [1, \ell]$, $\mathcal{P}$ computes vectors of degree-$(m-1)$ polynomials $\boldsymbol{f}_j(\cdot)$ and $\boldsymbol{g}_j(\cdot)$, such that $\boldsymbol{f}_j(i) = \boldsymbol{x}_{i,j}$ and $\boldsymbol{g}_j(i) = \boldsymbol{y}_{i,j}$ for all $i \in [1, m]$.
2. For $j \in [1, \ell]$, all verifiers locally compute $[\boldsymbol{f}_j(\cdot)]$ and $[\boldsymbol{g}_j(\cdot)]$ using their shares of $\{[\boldsymbol{x}_{i,j}]\}_{i \in [1,m]}$ and $\{[\boldsymbol{y}_{i,j}]\}_{i \in [1,m]}$ respectively.
3. For $i \in [m+1, 2m-1]$, $\mathcal{P}$ and all verifiers respectively compute the secrets and shares of packed sharings $[\boldsymbol{f}_j(i)]$ and $[\boldsymbol{g}_j(i)]$ for $j \in [1, \ell]$. Then, for $i \in [m+1, 2m-1]$, they execute the sub-protocol $\Pi^{\mathsf{pss}}_{\mathsf{inner\text{-}prod}}$ (shown in Figure 9) on $([\boldsymbol{f}_1(i)], \ldots, [\boldsymbol{f}_\ell(i)])$ and $([\boldsymbol{g}_1(i)], \ldots, [\boldsymbol{g}_\ell(i)])$ to compute the whole sharing $[\boldsymbol{z}_i] = [\sum_{j \in [1,\ell]} \boldsymbol{f}_j(i) * \boldsymbol{g}_j(i)]$, where $\mathcal{P}$ obtains $\boldsymbol{z}_i$ and the verifiers get the shares of $[\boldsymbol{z}_i]$. Besides, $\mathcal{P}$ and the verifiers obtain $\boldsymbol{u}_{m+1}, \ldots, \boldsymbol{u}_{2m-1} \in \mathbb{K}^k$.
4. $\mathcal{P}$ and all verifiers locally compute the whole sharing $[\boldsymbol{h}(\cdot)]$ from $[\boldsymbol{z}_1], \ldots, [\boldsymbol{z}_{2m-1}]$, such that $\boldsymbol{h}(\cdot)$ is a vector of degree-$2(m-1)$ polynomial and $\boldsymbol{h}(i) = \boldsymbol{z}_i$ for $i \in [1, 2m-1]$.
5. $\mathcal{P}$ and all verifiers compute $\alpha := \mathsf{H}_2(\gamma, \boldsymbol{v}_1, \ldots, \boldsymbol{v}_d, \boldsymbol{u}_{m+1}, \ldots, \boldsymbol{u}_{2m-1}) \in \mathbb{K}$. If $\alpha \in [1, m]$, the verifiers abort.
6. $\mathcal{P}$ and all verifiers output the secrets and shares of $([\boldsymbol{f}_1(\alpha)], \ldots, [\boldsymbol{f}_\ell(\alpha)])$, $([\boldsymbol{g}_1(\alpha)], \ldots, [\boldsymbol{g}_\ell(\alpha)])$ and $[\boldsymbol{h}(\alpha)]$ respectively, and also output the element α.

Fig. 10: **Protocol for compressing packed inner-product tuples.**

defined. In particular, χ can be defined as $\mathsf{H}_2(com_1, \ldots, com_n)$ as shown in Figure 7. When using $\Pi^{\mathsf{pss}}_{\mathsf{verifyprod}}$ to realize functionality $\mathcal{F}^{\mathsf{pss}}_{\mathsf{verifyprod}}$, $\chi \in \mathbb{K}$ can be generated by $\mathcal{P}$ and all verifiers in the main NIMVZK protocol shown in Figures 6 and 7. For the sake of simplicity, we ignore the case that the adversary (who corrupts $\mathcal{P}$) did not make a query to obtain χ but made a query to get a challenge $\alpha = \mathsf{H}_2(\chi, \cdots)$ used in protocol $\Pi^{\mathsf{pss}}_{\mathsf{verifyprod}}$, which occurs with probability at most $\frac{1}{|\mathbb{K}|} \leq \frac{1}{2^\lambda}$. When the adversary makes a query $(com_1, \ldots, com_n)$ to random oracle H_2 and obtains χ, the challenge χ is determined after the secrets stored in the input packed tuple have been defined, except with probability at most $\frac{Q_1 n}{2^\lambda}$ following the proof of Theorem 3.

Sub-protocol for computing inner product. Sub-protocol $\Pi^{\mathsf{pss}}_{\mathsf{inner\text{-}prod}}$ shown in Figure 9 is used to compute the inner product of two vectors $([\boldsymbol{x}_1], \ldots, [\boldsymbol{x}_\ell])$ and $([\boldsymbol{y}_1], \ldots, [\boldsymbol{y}_\ell])$. The prover now knows all the challenges due to the use of the Fiat-Shamir transform, and thus holds all the secrets involved in the verification procedure. Thus, the prover can directly distribute the shares of $[\boldsymbol{z}]$ with $\boldsymbol{z} = \sum_{h \in [1,\ell]} \boldsymbol{x}_h * \boldsymbol{y}_h$ to all verifiers. To support Fiat-Shamir, the verifiers need to know public messages instead of secret shares. Therefore, we first let the prover generate a random packed sharing $[\boldsymbol{r}]$, and then make it broadcast the *public* difference $\boldsymbol{u} = \boldsymbol{z} + \boldsymbol{r}$ to all verifiers. The prover and verifiers also need to output the message $\boldsymbol{u}$, which will be used in the Fiat-Shamir transform of the main verification protocol. When the prover is malicious, it can introduce an additive error to the sharing $[\boldsymbol{z}]$ output by the verifiers, which is harmless when integrating $\Pi^{\mathsf{pss}}_{\mathsf{inner\text{-}prod}}$ into the main protocol $\Pi^{\mathsf{pss}}_{\mathsf{verifyprod}}$.

Sub-protocol for compression. Sub-protocol $\Pi^{\mathsf{pss}}_{\mathsf{compress}}$ shown in Figure 10 is used to compress m packed inner-product tuples into a single packed inner-product tuple. In particular, this protocol invokes the sub-protocol $\Pi^{\mathsf{pss}}_{\mathsf{inner\text{-}prod}}$ instead of calling an inner-product functionality, which seems necessary to support Fiat-Shamir, where the messages related to the secrets need to be used as the input of a random oracle H_2. For every protocol execution of $\Pi^{\mathsf{pss}}_{\mathsf{compress}}$, the prover and all verifiers generate a random challenge $\alpha \in \mathbb{K}$ using the Fiat-Shamir transform. To realize *non-interactively* recursive compression in the main verification protocol, the prover and verifiers also input the public challenge γ from the previous iteration and the public messages produced in the current iteration.

Theorem 4. *Let $\mathsf{H}_2 : \{0,1\}^* \to \mathbb{K}$ be a random oracle. Protocol $\Pi^{\mathsf{pss}}_{\mathsf{verifyprod}}$ shown in Figure 8 securely realizes functionality $\mathcal{F}^{\mathsf{pss}}_{\mathsf{verifyprod}}$ with soundness error at most $\frac{4\lceil \log M \rceil + 5Q_2}{2^\lambda - 3}$ in the presence of a malicious adversary corrupting up to the prover and exactly t verifiers, where Q_2 is the number of queries to random oracle H_2.*

The proof of Theorem 4 is given in the full version [52].

Acknowledgements

Work of Kang Yang is supported by the National Natural Science Foundation of China (Grant Nos. 62102037, 61932019). Work of Xiao Wang is supported in part by DARPA under Contract No. HR001120C0087, NSF award #2016240, and research awards from Meta, Google and PlatON Network. The views, opinions, and/or findings expressed are those of the author(s) and should not be

interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

References

1. Abe, M.: Robust Distributed Multiplication Without Interaction. In: Advances in Cryptology—Crypto 1999. LNCS, vol. 1666, pp. 130–147. Springer (1999). https://doi.org/10.1007/3-540-48405-1_9
2. Abe, M., Cramer, R., Fehr, S.: Non-interactive Distributed-Verifier Proofs and Proving Relations among Commitments. In: Advances in Cryptology—Asiacrypt 2002. pp. 206–223. LNCS, Springer (2002). https://doi.org/10.1007/3-540-36178-2_13
3. Ames, S., Hazay, C., Ishai, Y., Venkitasubramaniam, M.: Ligero: Lightweight Sublinear Arguments Without a Trusted Setup. In: ACM Conf. on Computer and Communications Security (CCS) 2017. pp. 2087–2104. ACM Press (2017). https://doi.org/10.1145/3133956.3134104
4. Applebaum, B., Kachlon, E., Patra, A.: Verifiable Relation Sharing and Multi-Verifier Zero-Knowledge in Two Rounds: Trading NIZKs with Honest Majority. Cryptology ePrint Archive, Paper 2022/167 (2022), https://eprint.iacr.org/2022/167
5. Baldimtsi, F., Kiayias, A., Zacharias, T., Zhang, B.: Crowd Verifiable Zero-Knowledge and End-to-End Verifiable Multiparty Computation. In: Advances in Cryptology—Asiacrypt 2020, Part III. pp. 717–748. LNCS, Springer (2020). https://doi.org/10.1007/978-3-030-64840-4_24
6. Baum, C., Braun, L., Munch-Hansen, A., Razet, B., Scholl, P.: Appenzeller to Brie: Efficient Zero-Knowledge Proofs for Mixed-Mode Arithmetic and Z2k. In: ACM Conf. on Computer and Communications Security (CCS) 2021. pp. 192–211. ACM Press (2021). https://doi.org/10.1145/3460120.3484812
7. Baum, C., Jadoul, R., Orsini, E., Scholl, P., Smart, N.P.: Feta: Efficient Threshold Designated-Verifier Zero-Knowledge Proofs. Cryptology ePrint Archive, Paper 2022/082 (2022), https://eprint.iacr.org/2022/082
8. Baum, C., Malozemoff, A.J., Rosen, M.B., Scholl, P.: Mac'n'Cheese: Zero-Knowledge Proofs for Boolean and Arithmetic Circuits with Nested Disjunctions. In: Advances in Cryptology—Crypto 2021, Part IV. pp. 92–122. LNCS, Springer (2021). https://doi.org/10.1007/978-3-030-84259-8_4
9. Beck, G., Goel, A., Jain, A., Kaptchuk, G.: Order-C Secure Multiparty Computation for Highly Repetitive Circuits. In: Advances in Cryptology—Eurocrypt 2021, Part II. pp. 663–693. LNCS, Springer (2021). https://doi.org/10.1007/978-3-030-77886-6_23
10. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable Zero Knowledge with No Trusted Setup. In: Advances in Cryptology—Crypto 2019, Part III. LNCS, vol. 11694, pp. 701–732. Springer (2019). https://doi.org/10.1007/978-3-030-26954-8_23
11. Ben-Sasson, E., Chiesa, A., Riabzev, M., Spooner, N., Virza, M., Ward, N.P.: Aurora: Transparent Succinct Arguments for R1CS. In: Advances in Cryptology—Eurocrypt 2019, Part I. LNCS, vol. 11476, pp. 103–128. Springer (2019). https://doi.org/10.1007/978-3-030-17653-2_4
12. Ben-Sasson, E., Chiesa, A., Spooner, N.: Interactive Oracle Proofs. In: 9th Theory of Cryptography Conference—TCC 2016. pp. 31–60. LNCS, Springer (2016). https://doi.org/10.1007/978-3-662-53644-5_2

13. Bhadauria, R., Fang, Z., Hazay, C., Venkitasubramaniam, M., Xie, T., Zhang, Y.: Ligero++: A New Optimized Sublinear IOP. In: ACM Conf. on Computer and Communications Security (CCS) 2020. pp. 2025–2038. ACM Press (2020). https://doi.org/10.1145/3372297.3417893
14. Boneh, D., Boyle, E., Corrigan-Gibbs, H., Gilboa, N., Ishai, Y.: Zero-Knowledge Proofs on Secret-Shared Data via Fully Linear PCPs. In: Advances in Cryptology—Crypto 2019, Part III. LNCS, vol. 11694, pp. 67–97. Springer (2019). https://doi.org/10.1007/978-3-030-26954-8_3
15. Bootle, J., Cerulli, A., Chaidos, P., Groth, J., Petit, C.: Efficient Zero-Knowledge Arguments for Arithmetic Circuits in the Discrete Log Setting. In: Advances in Cryptology—Eurocrypt 2016, Part II. LNCS, vol. 9666, pp. 327–357. Springer (2016). https://doi.org/10.1007/978-3-662-49896-5_12
16. Boyle, E., Gilboa, N., Ishai, Y., Nof, A.: Efficient Fully Secure Computation via Distributed Zero-Knowledge Proofs. In: Advances in Cryptology—Asiacrypt 2020, Part III. pp. 244–276. LNCS, Springer (2020). https://doi.org/10.1007/978-3-030-64840-4_9
17. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short Proofs for Confidential Transactions and More. In: IEEE Symp. Security and Privacy 2018. pp. 315–334. IEEE (2018). https://doi.org/10.1109/SP.2018.00020
18. Bünz, B., Fisch, B., Szepieniec, A.: Transparent SNARKs from DARK Compilers. In: Advances in Cryptology—Eurocrypt 2020, Part I. LNCS, vol. 12105, pp. 677–706. Springer (2020). https://doi.org/10.1007/978-3-030-45721-1_24
19. Burmester, M., Desmedt, Y.: Broadcast Interactive Proofs (Extended Abstract). In: Advances in Cryptology—Eurocrypt 1991. pp. 81–95. LNCS, Springer (1991). https://doi.org/10.1007/3-540-46416-6_7
20. Canetti, R., Chen, Y., Holmgren, J., Lombardi, A., Rothblum, G.N., Rothblum, R.D., Wichs, D.: Fiat-Shamir: from practice to theory. In: 51th Annual ACM Symposium on Theory of Computing (STOC). pp. 1082–1090. ACM Press (2019). https://doi.org/10.1145/3313276.3316380
21. Canetti, R., Kaptchuk, G.: The Broken Promise of Apple's Announced Forbidden-photo Reporting System – And How To Fix It. https://www.bu.edu/riscs/2021/08/10/apple-csam/ (2021)
22. Chida, K., Genkin, D., Hamada, K., Ikarashi, D., Kikuchi, R., Lindell, Y., Nof, A.: Fast Large-Scale Honest-Majority MPC for Malicious Adversaries. In: Advances in Cryptology—Crypto 2018, Part III. LNCS, vol. 10993, pp. 34–64. Springer (2018). https://doi.org/10.1007/978-3-319-96878-0_2
23. Corrigan-Gibbs, H., Boneh, D.: Prio: Private, Robust, and Scalable Computation of Aggregate Statistics. In: 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17). pp. 259–282. USENIX Association (Mar 2017)
24. Damgård, I., Ishai, Y., Krøigaard, M.: Perfectly Secure Multiparty Computation and the Computational Overhead of Cryptography. In: Advances in Cryptology—Eurocrypt 2010. pp. 445–465. LNCS, Springer (2010). https://doi.org/10.1007/978-3-642-13190-5_23
25. Damgård, I., Nielsen, J.B.: Scalable and Unconditionally Secure Multiparty Computation. In: Advances in Cryptology—Crypto 2007. LNCS, vol. 4622, pp. 572–590. Springer (2007). https://doi.org/10.1007/978-3-540-74143-5_32
26. Damgård, I., Nielsen, J.B., Nielsen, M., Ranellucci, S.: The TinyTable Protocol for 2-Party Secure Computation, or: Gate-Scrambling Revisited. In: Advances in Cryptology—Crypto 2017, Part I. LNCS, vol. 10401, pp. 167–187. Springer (2017). https://doi.org/10.1007/978-3-319-63688-7_6

27. Dittmer, S., Ishai, Y., Lu, S., Ostrovsky, R.: Improving Line-Point Zero Knowledge: Two Multiplications for the Price of One. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. ACM Press (2022)
28. Dittmer, S., Ishai, Y., Ostrovsky, R.: Line-Point Zero Knowledge and Its Applications. In: 2nd Conference on Information-Theoretic Cryptography (2021)
29. Fiat, A., Shamir, A.: How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In: Advances in Cryptology—Crypto 1986. pp. 186–194. LNCS, Springer (1987). https://doi.org/10.1007/3-540-47721-7_12
30. Franklin, M.K., Yung, M.: Communication Complexity of Secure Computation (Extended Abstract). In: 24th Annual ACM Symposium on Theory of Computing (STOC). pp. 699–710. ACM Press (1992). https://doi.org/10.1145/129712.129780
31. Garay, J.A., Ishai, Y., Ostrovsky, R., Zikas, V.: The Price of Low Communication in Secure Multi-party Computation. In: Advances in Cryptology—Crypto 2017, Part I. LNCS, vol. 10401, pp. 420–446. Springer (2017). https://doi.org/10.1007/978-3-319-63688-7_14
32. Genkin, D., Ishai, Y., Polychroniadou, A.: Efficient Multi-party Computation: From Passive to Active Security via Secure SIMD Circuits. In: Advances in Cryptology—Crypto 2015, Part II. LNCS, vol. 9216, pp. 721–741. Springer (2015). https://doi.org/10.1007/978-3-662-48000-7_35
33. Genkin, D., Ishai, Y., Prabhakaran, M., Sahai, A., Tromer, E.: Circuits resilient to additive attacks with applications to secure computation. In: 46th Annual ACM Symposium on Theory of Computing (STOC). pp. 495–504. ACM Press (2014). https://doi.org/10.1145/2591796.2591861
34. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic Span Programs and Succinct NIZKs without PCPs. In: Advances in Cryptology—Eurocrypt 2013. pp. 626–645. LNCS, Springer (2013). https://doi.org/10.1007/978-3-642-38348-9_37
35. Goldwasser, S., Kalai, Y.T., Rothblum, G.N.: Delegating computation: interactive proofs for muggles. In: 40th Annual ACM Symposium on Theory of Computing (STOC). pp. 113–122. ACM Press (2008). https://doi.org/10.1145/1374376.1374396
36. Goldwasser, S., Lindell, Y.: Secure Multi-Party Computation without Agreement. J. Cryptology **18**(3), 247–287 (Jul 2005). https://doi.org/10.1007/s00145-005-0319-z
37. Gordon, S.D., Starin, D., Yerukhimovich, A.: The More the Merrier: Reducing the Cost of Large Scale MPC. In: Advances in Cryptology—Eurocrypt 2021, Part II. pp. 694–723. LNCS, Springer (2021). https://doi.org/10.1007/978-3-030-77886-6_24
38. Goyal, V., Li, H., Ostrovsky, R., Polychroniadou, A., Song, Y.: ATLAS: Efficient and Scalable MPC in the Honest Majority Setting. In: Advances in Cryptology—Crypto 2021, Part II. pp. 244–274. LNCS, Springer (2021). https://doi.org/10.1007/978-3-030-84245-1_9
39. Goyal, V., Polychroniadou, A., Song, Y.: Unconditional Communication-Efficient MPC via Hall's Marriage Theorem. In: Advances in Cryptology—Crypto 2021, Part II. pp. 275–304. LNCS, Springer (2021). https://doi.org/10.1007/978-3-030-84245-1_10
40. Goyal, V., Song, Y.: Malicious Security Comes Free in Honest-Majority MPC. Cryptology ePrint Archive, Report 2020/134 (2020), https://eprint.iacr.org/2020/134

41. Goyal, V., Song, Y., Zhu, C.: Guaranteed Output Delivery Comes Free in Honest Majority MPC. In: Advances in Cryptology—Crypto 2020, Part II. pp. 618–646. LNCS, Springer (2020). https://doi.org/10.1007/978-3-030-56880-1_22
42. Groth, J.: Short Pairing-Based Non-interactive Zero-Knowledge Arguments. In: Advances in Cryptology—Asiacrypt 2010. pp. 321–340. LNCS, Springer (2010). https://doi.org/10.1007/978-3-642-17373-8_19
43. Groth, J., Ostrovsky, R.: Cryptography in the Multi-string Model. In: Advances in Cryptology—Crypto 2007. LNCS, vol. 4622, pp. 323–341. Springer (2007). https://doi.org/10.1007/978-3-540-74143-5_18
44. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Zero-knowledge from secure multiparty computation. In: 39th Annual ACM Symposium on Theory of Computing (STOC). pp. 21–30. ACM Press (2007). https://doi.org/10.1145/1250790.1250794
45. Lepinski, M., Micali, S., shelat, a.: Fair-Zero Knowledge. In: 2nd Theory of Cryptography Conference—TCC 2005. LNCS, vol. 3378, pp. 245–263. Springer (2005). https://doi.org/10.1007/978-3-540-30576-7_14
46. Lindell, Y., Nof, A.: A Framework for Constructing Fast MPC over Arithmetic Circuits with Malicious Adversaries and an Honest-Majority. In: ACM Conf. on Computer and Communications Security (CCS) 2017. pp. 259–276. ACM Press (2017). https://doi.org/10.1145/3133956.3133999
47. Nordholt, P.S., Veeningen, M.: Minimising Communication in Honest-Majority MPC by Batchwise Multiplication Verification. In: Intl. Conference on Applied Cryptography and Network Security (ACNS). pp. 321–339. LNCS, Springer (2018). https://doi.org/10.1007/978-3-319-93387-0_17
48. Setty, S.: Spartan: Efficient and General-Purpose zkSNARKs Without Trusted Setup. In: Advances in Cryptology—Crypto 2020, Part III. pp. 704–737. LNCS, Springer (2020). https://doi.org/10.1007/978-3-030-56877-1_25
49. Wahby, R.S., Tzialla, I., shelat, a., Thaler, J., Walfish, M.: Doubly-Efficient zk-SNARKs Without Trusted Setup. In: IEEE Symp. Security and Privacy 2018. pp. 926–943. IEEE (2018). https://doi.org/10.1109/SP.2018.00060
50. Weng, C., Yang, K., Katz, J., Wang, X.: Wolverine: Fast, Scalable, and Communication-Efficient Zero-Knowledge Proofs for Boolean and Arithmetic Circuits. In: IEEE Symp. Security and Privacy 2021. pp. 1074–1091. IEEE (2021). https://doi.org/10.1109/SP40001.2021.00056
51. Yang, K., Sarkar, P., Weng, C., Wang, X.: QuickSilver: Efficient and Affordable Zero-Knowledge Proofs for Circuits and Polynomials over Any Field. In: ACM Conf. on Computer and Communications Security (CCS) 2021. pp. 2986–3001. ACM Press (2021). https://doi.org/10.1145/3460120.3484556
52. Yang, K., Wang, X.: Non-Interactive Zero-Knowledge Proofs to Multiple Verifiers. Cryptology ePrint Archive, Paper 2022/063 (2022), https://eprint.iacr.org/2022/063
53. Zhang, J., Liu, T., Wang, W., Zhang, Y., Song, D., Xie, X., Zhang, Y.: Doubly Efficient Interactive Proofs for General Arithmetic Circuits with Linear Prover Time. In: ACM Conf. on Computer and Communications Security (CCS) 2021. pp. 159–177. ACM Press (2021). https://doi.org/10.1145/3460120.3484767
54. Zhang, J., Xie, T., Zhang, Y., Song, D.: Transparent Polynomial Delegation and Its Applications to Zero Knowledge Proof. In: IEEE Symp. Security and Privacy 2020. pp. 859–876. IEEE (2020). https://doi.org/10.1109/SP40000.2020.00052