Crowd Verifiable Zero-Knowledge and End-to-end Verifiable Multiparty Computation

Foteini Baldimtsi[1], Aggelos Kiayias[2][3], Thomas Zacharias[2], and Bingsheng Zhang[4]

[1] George Mason University, USA
[2] The University of Edinburgh, UK
[3] IOHK, UK
[4] Alibaba-Zhejiang University Joint Research Institute of Frontier Technologies
Zhejiang University, China

Abstract. Auditing a secure multiparty computation (MPC) protocol entails the validation of the protocol transcript by a third party that is otherwise untrusted. In this work we introduce the concept of *end-to-end verifiable* MPC (VMPC), that requires the validation to provide a correctness guarantee even in the setting that all servers, trusted setup primitives and all the client systems utilized by the input-providing users of the MPC protocol are subverted by an adversary. To instantiate VMPC, we introduce a new concept in the setting of zero-knowlegde protocols that we term *crowd verifiable zero-knowledge* (CVZK). A CVZK protocol enables a prover to convince a set of verifiers about a certain statement, even though each one individually contributes a small amount of entropy for verification and some of them are adversarially controlled. Given CVZK, we present a VMPC protocol that is based on discrete-logarithm related assumptions. At the high level of adversity that VMPC is meant to withstand, it is infeasible to ensure perfect correctness, thus we investigate the classes of functions and verifiability relations that are feasible in our framework, and present a number of possible applications the underlying functions of which can be implemented via VMPC.

Keywords: Multi-party computation· zero-knowledge· privacy· verifiability

1 Introduction

Over the last 30 years, secure multiparty computation (MPC) has transitioned from theoretical feasibility results [62, 63, 35] to real-world implementations [13, 60, 46, 29, 28, 26] that can be used for a number of different security critical operations including auctions [13], e-voting [25, 44, 2], and privacy preserving statistics [53, 14]. An important paradigm for MPC that captures a large number of applications is the *client-server* model [32, 7, 27, 41, 54, 36] where participants of the system are distinguished between clients and servers, with the clients contributing input for the computation and receiving the output, while the servers, operating in an oblivious fashion, are processing the data given by the clients.

The servers performing the MPC protocol collectively ensure the privacy preservation of the execution, up to the information that is leaked by the output itself. There do exist protocols that achieve this level of privacy provided that there exists *at least one server* that is not subverted by the adversary. The typical execution of such protocols involves the clients encoding their input suitably for processing by the servers (e.g., by performing secret-sharing [38]) and receiving the encoded output which they reconstruct to produce the final result. While the level of privacy achieved by such protocols is adequate for their intended applications and their performance has improved over time (e.g., protocols such as SPDZ [29] and [28, 42] achieve very good performance for real world applications by utilizing an offline/online approach [6]), there are still crucial considerations for their deployment in the real-world especially if the outcome of the MPC protocol has important committing and actionable consequences (such as e.g., in e-voting, auctions and other protocols).

To address this consideration, Baum, Damgård and Orlandi [4] asked whether it is feasible to construct efficient *auditable* MPC protocols. In auditable MPC, an external observer who is given access to the protocol transcript, can verify that the protocol was executed correctly even if all the servers (but not client devices) were subverted by the adversary. The authors of [4] observe that this is theoretically feasible if a common reference string (CRS) is available to the participants and provide an efficient instantiation of such protocol by suitably amending the SPDZ protocol [29]. While the above constitutes a good step towards addressing real world considerations of deploying MPC protocols, there are serious issues that remain from the perspective of auditability. Specifically, the work of [4] does not provide any guarantees about the validity of the output in case, (i) the CRS is subverted, or (ii) the users' client devices get corrupted.

Verification of the correctness of the result by any party, even if all servers are corrupt (but not client devices), has also been studied by Schoenmakers and Veeningen [61] in the context of *universally verifiable* MPC. The security analysis in [61] is in the random oracle model and still, the case of corrupted client devices is not considered. Moreover, achieving universally verifiable (or publicly auditable) MPC in the standard model is stated as an open problem.

Unfortunately, the threat of malicious CRS and client byzantine behavior cannot be dismissed: in fact, it has been extensively studied in the context of e-voting systems, which are a very compelling use-case for MPC, and frequently invoked as one of the important considerations for real-world deployment. Specifically, the issue of malicious clients has been studied in the end-to-end verifiability model for e-voting, e.g., [47] while the issue of removing setup assumptions such as the CRS or random oracles has been also recently considered [44, 43].

The fact that the concept of end-to-end verifiability has been so far thoroughly examined in the e-voting area comes not as surprise, since elections is a prominent example where auditing the correctness of the execution is a top integrity requirement. Nonetheless, transparency in terms of end-to-end verification can be a highly desirable feature in several other scenarios, such as auctions, demographic statistics, financial analysis, or profile matching where the (human) users contributing their inputs may have a keen interest in auditing the correctness of the computation (e.g., highest bid, unemployment rate, average salary, order book matching in trading). From a mathematical aspect, it appears that several other use-cases of MPC evaluation functions besides tallying that fall into the scope of end-to-end verification have not been examined.

To capture these considerations and instead of pursuing tailored-made studies for each use-case, in this work, we take a step forward and propose a unified treatment of the problem of end-to-end verifiability in MPC under a "human-client-server" setting. In particular, we separate human users from their client devices (e.g., smartphones) in the spirit of the "ceremony" concept [31, 45] of voting protocols. While client devices can be thought of as stateful, probabilistic, interactive Turing machines, we model human users to be limited in two ways: (a) humans are bad sources of randomness; formally, the randomness of a user can be adversarially guessed with non-negligible probability, i.e. its min-entropy is up to logarithmic to the security parameter, and (b) humans cannot perform complicated calculations; i.e. humans' computational complexity is linear in the security parameter (i.e., the minimum for reading the input). Given this modeling we ask:

> *Is it possible to construct auditable MPC protocols, in the sense that everyone who has access to the transcript can verify that the output is correct, even if* all *servers, client devices and setup assumptions (e.g. a common reference string) are subverted by an adversary?*

We answer this question by introducing the concept of *end-to-end verifiable multiparty computation* (VMPC) and presenting both feasibility and infeasibility results for different classes of functions. Some of the most promising applications of VMPC include e-voting, privacy preserving statistics and supervised learning of classifiers over private data.

1.1 Technical Overview and Contributions

VMPC model. The security property of VMPC is modeled in the universal composability (UC) framework [16], aiming to unifying two lines of research on secure computing: end-to-end verifiable e-voting (which typically separates humans from their devices in security analysis) and client-server (auditable) MPC. More specifically, we define the VMPC ideal functionality as $\mathcal{F}_{\mathsf{vmpc}}^{f,R}(\mathcal{P})$, where $\mathcal{P}$ is a set of players, including users, client devices, servers and a verifier; f is the MPC function to be evaluated, and R is a relation that is used to measure the distance between the returned VMPC output and the correct (true) computation result. As will be explained later, when the VMPC output is verified, it is guaranteed that the output is not "far" from the truth.

The distinction between users and clients. In order to capture "end-to-end verifiability", we have to make a distinction between users and clients: the users are the humans with limited computation and entropy that interact with their client devices (e.g., smartphones or laptops) to provide input to the MPC. To accommodate this, our ideal functionality acknowledges these two roles and for this reason it departs from the previous formulation of auditable MPC [4]. A critical challenge in VMPC is the fact that the result should be verifiable even if *all* clients and servers are corrupted!

The role of the verifier. VMPC departs from the conventional UC definition of MPC since there should be a special entity, the verifier, that verifies the correctness of the output. The concept of the verifier in our modeling is an abstraction only. The verifier is invoked only for auditing and trusted only for verifiability, not privacy. It can be any device, organization, or computer system that the user trusts to do the audit. Moreover, it is straightforward to extend the model to involve multiple verifiers as discussed in Section 5 and hence only for simplicity we choose to model just a single entity. We note that the human user cannot perform auditing herself due to the fact that it requires cryptographic computations. As in e-voting, verification is delegatable, i.e., the verifier obtains users' individual audit data in an out-of-band manner.

EUC with a super-polynomial helper. The astute readers may notice that a UC realization of the VMPC primitive in a setting where there is no trusted setup such as a CRS is infeasible. Indeed, it is well known [16] that a non-trivial MPC functionalities cannot be UC-realized without a trusted setup. To go around these impossibility results and still provide a composable construction, we utilize the extended UC model with a helper $\mathcal{H}$, ($\mathcal{H}$-EUC security) [18]. This model, which can been seen as an adaptation of the super-polynomial simulation concept [59] in the UC setting, enables one to provide standard model constructions that are composable and at the same time real world secure, using a "complexity leveraging" argument that requires subexponential security for the underlying cryptographic primitives. In particular, in the setting of $\mathcal{H}$-EUC security, translating a real world attack to an ideal world attack requires a super-polynomial computation. More precisely, a polynomial-time operation that invokes a super-polynomial helper program $\mathcal{H}$. It follows that if the distance of the real world from the ideal is bounded by the distinguishing advantage of some underlying cryptographic distributions, assuming subexponential indistinguishability is sufficient to infer the security for the primitive.

System architecture. We assume there exists a *consistent and public bulletin board (BB)* (modeled as the global functionality $\mathcal{G}_{\mathsf{BB}}$) that can be accessed by all the VMPC players except human users, i.e., the client devices, the servers and the verifier. In addition, we assume there exists an authenticated channel (modeled as the functionality $\mathcal{F}_{\mathsf{auth}}$) between the human users and the verifier. Besides, we assume there exists a secure channel (modeled as the functionality $\mathcal{F}_{\mathsf{sc}}$) between the human users and their local client devices. A VMPC scheme consists of four sub-protocols: Initialize (setup phase among servers), Input (run by servers, users-clients), Compute (executed by the servers) and Verify (executed by the verifier and users). According to the e-voting and pre-processing MPC approach [12, 57, 29, 28], we consider *minimal user interaction* - the users independently interact with the system once in order to submit their inputs. This limitation is challenging from a protocol design perspective.

The breadth of VMPC feasibility. We explore the class of functions that can be realized by VMPC, since in our setting, contrary to general MPC results, it is *infeasible* to compute any function with perfect correctness. To see this with a simple example, consider some function f that outputs the XOR of the input bits. It is easy to see that each user has too little entropy to challenge the set of malicious clients and servers about the proper encoding of her private input. However, even if a single input bit is incorrectly encoded by the user's client (which can be undetected with non-negligible probability) the output XOR value can be flipped. To accommodate for this natural deficiency, our VMPC functionality enforces a relation R between the reported output and the correct output. It is clear that depending on the function f, a different relation R may be achievable. We capture this interplay between correctness and the function to be computed by introducing the notion of a *spreading relation* R for a function $f : X \to Y$. Informally, given a certain metric over the input space, a spreading relation over the range of f, satisfies that whenever x, x' are close w.r.t. the metric, the images of x, x' are related. A typical case of a spreading relation can emerge when f is a Lipschitz function for a given metric. Based on the above we show that one cannot hope to compute a function f with a relation over the range of f that is more "refined" than a spreading relation.

Building Blocks. VMPC is a complex primitive and we introduce *novel building blocks* to facilitate it. ZK proofs cannot be directly used for VMPC since we require a 3-round public-coin protocol to comply with our minimal interaction setting and this is infeasible, cf. [40, 33], while we cannot utilize a subversion-sound NIZK either, cf. [8], since in this case, we can at best obtain witness indistinguishability which is insufficient for proving the simulation-based privacy needed for VMPC.

Crowd Verifiable Zero-Knowledge (CVZK). To overcome these issues we introduce a new cryptographic primitive that we call *crowd verifiable zero-knowledge* which may also be of independent interest. In CVZK, a single prover tries to convince a set of n verifiers (a "crowd") of the validity of a certain statement. Although the notion of multi-verifier zero-knowledge already exists in the literature, e.g. [15, 50], the focus of CVZK is different. Namely, the challenge for CVZK is that each human verifier is restricted to contribute up to a logarithmic number of random bits and hence, if, say all but one verifiers are corrupted, there would be insufficient entropy available in order to achieve a low soundness error. Thus, the only way to go forward for the verifiers is to assume the relative honesty of the crowd, i.e., there is a sufficient number of them acting honestly and introduce enough randomness in the system so that the soundness error can be small. The notion of CVZK is critical towards realizing VMPC, since in the absence of reliable client systems, the users have no obvious way of challenging the system's operation; users, being humans, are assumed to be bad sources of entropy that cannot contribute individually a sufficient number of random bits to provide a sufficiently low soundness error.

Coalescence functions and CVZK Instantiation. We introduce *coalescence functions* (Section 3.2) to typify the randomness extraction primitive that is at the core of our CVZK construction. In CVZK, it is not straightforward how to use the random bits that honest verifiers contribute. The reason is that the adversary, who is in control of the prover and a number of verifiers, may attempt to use the malicious verifiers' coins to "cancel" the entropy of the honest verifiers and assist the malicious prover to convince them of a wrong statement. Coalescence relates to collective coin flipping [9] and randomness condensers [30]. In particular, a coalescence function is a deterministic function that tries to make good use of the entropy of its input. Specifically, a coalescence function takes as an input a non-oblivious symbol fixing source and produces a series of blocks, one of which is guaranteed to be of high entropy; these blocks will be subsequently used in conjunction to form the challenge implementing CVZK.

We construct coalescence functions using a one-round collective coin flipping protocol and the (strongly) resilient function defined in [55]. Then, we present a compiler that takes a fully input delayed Σ-protocol and leads to a CVZK construction that performs a parallel proof w.r.t. each block produced by the coalescence function. Our CVZK construction is secure for any number of corrupted users up to $O(n^c/\log^3 n)$, for some constant $c < 1$ and a set of n users.

VMPC Construction. Our VMPC construction is based on CVZK. It uses an offline / online approach (a.k.a. pre-processing mode) for computing the output (proposed by Beaver [6] and utilized numerous times [29, 4]). In a nutshell, our construction follows the paradigm of SPDZ [29] and BDO [4]. Namely, the data are shared and committed on the BB. The underlying secret sharing scheme and the commitment scheme have compatible linearly homomorphic property; therefore, the auditor can check the correctness of the protocol execution by performing the same operations over the committed data. In addition, to achieve crowd verifiability, all the ZK proofs need to be transformed to CVZK – (i) in the pre-processing phase, the servers post the first move of the CVZK on the BB; (ii) in the input phase, the (human) users collaboratively generate the challenge coins of the CVZK; (iii) in the output phase, the servers post the protocol output together with the third move of the CVZK, which completes the CVZK proofs.

We prove indistinguishability between real and ideal world for our construction under adaptive onewayness [58] of the discrete-logarithm function and the decisional Diffie-Hellman assumption. We infer that, by utilizing sub-exponential versions of those assumptions, our protocol realizes the ideal description of VMPC, in the $\mathcal{H}$-EUC model, for any (symmetric) function f with correctness up to a spreading relation R for f.

We note that an alternative but sub-optimal approach to VMPC would be to add the Benaloh challenge mechanism [10, 11], that has been proposed in the context of e-voting to mitigate corrupted client devices, to the BDO protocol [4]. However, the resulting VMPC protocol would still require a trusted setup, e.g., CRS or Random Oracle (RO), and therefore it would fall short of our objective to realize VMPC in the plain model. Moreover, the Benaloh challenge mechanism requires the client to have a second trusted device that is capable of performing a cryptographic computation *prior to submitting her input* to the VMPC protocol and being able to communicate with it in an authenticated manner. Instead, the only requirement in our VMPC protocol is to have authenticated access to a verifier in the final step of the protocol.

Applications. As already mentioned, a main motivation for this work is the apparent connection of end-to-end verifiability to several practical MPC instantiations for real-world scenarios. Thus, we conclude by discussing possible applications of VMPC and examine how their underlying function can be combined with suitable spreading relations and implemented. We provide some interesting examples: (i) E-voting functions: where the final election tally aggregates the votes provided by the voters, (ii) privacy-preserving statistics: where the final outcome is a statistic that is calculated over uni-dimensional data, (iii) privacy-preserving processing of multi-dimensional data: where functions that correlate across different dimensions are calculated, (iv) supervised learning of classifiers: where the outcome is a model that results from training on private data.

2 Preliminaries

Notation. By λ we denote the security parameter and by $\mathsf{negl}(\cdot)$ the property that a function is negligible in some parameter. We write $\mathrm{poly}(x)$ to denote that a value is polynomial in x, PPT to denote probabilistic polynomial time, and $[n]$ as the abbreviation of the set $\{1, \ldots, n\}$. $H_{\min}(\mathbb{D})$ denotes the min entropy of a distribution $\mathbb{D}$ and $\mathbb{U}_n$ denotes the uniform distribution over $\{0,1\}^n$. By $x \overset{\$}{\leftarrow} S$, we denote that x is sampled uniformly at random from set S, and by $X \sim \mathbb{D}$ that the random variable X follows the distribution $\mathbb{D}$.

***Σ*-protocols.** Let $R_{\mathcal{L}}$ be polynomial-time-decidable witness relation for an **NP**-language $\mathcal{L}$. A *Σ-protocol* is a 3-move public coin protocol between a prover, $\Sigma.\mathsf{Prv}$, and a verifier, $\Sigma.V$, where the goal of the prover, having a witness w, is to convince the verifier that some statement x is in language $\mathcal{L}$. We split the prover $\Sigma.\mathsf{Prv}$ into two algorithms $(\Sigma.\mathsf{Prv}_1, \Sigma.\mathsf{Prv}_2)$. A Σ-protocol for $(x, w) \in \mathcal{R}_{\mathcal{L}}$ consists of the following PPT algorithms:

- $\Sigma.\mathsf{Prv}_1(x, w)$: on input $x \in \mathcal{L}$ and w s.t. $(x, w) \in \mathcal{R}_{\mathcal{L}}$, it outputs the first message of the protocol, a, and a state $\mathsf{st}_P \in \{0,1\}^*$.
- $\Sigma.\mathsf{Prv}_2(\mathsf{st}_P, e)$: after receiving the challenge $e \in \{0,1\}^\lambda$ from $\Sigma.V$ and on input the state, it outputs the prover's response z.
- $\Sigma.\mathsf{Verify}(x, a, e, z)$: on input a transcript (x, a, e, z), it outputs $b \in \{0,1\}$. A transcript is called *accepting* if $\Sigma.\mathsf{Verify}(x, a, e, z) = 1$.

We care about the following properties: (i) completeness, (ii) special soundness, and (iii) *special honest verifier zero-knowledge (sHVZK)*, i.e., if the challenge e is known in advance, then there is a PPT simulator $\Sigma.\mathsf{Sim}$ that simulates the transcript on input (x, e). In addition, we allow completeness of a Σ-protocol to be non-perfect, i.e. have a negligible error, and sHVZK to be computational. We provide a formal definition of a Σ-protocol in Supplementary Material A.1.

One-round collective coin flipping and resilient functions. The core of our CVZK construction is similar to a *one-round collective coin flipping (1RCCF)* process: (1) each player generates and broadcasts a coin c within the same round, (2) a uniformly random string is produced (with high probability).

The adversary can see the honest players' coins first and then decide the corrupted players' coins. The 1RCCF notion was introduced in [9] and is closely related to the notion of resilient functions which we recall below.

Definition 1 (Resilient function). *Let $f : \{0,1\}^m \longrightarrow \{0,1\}$ be a Boolean function on variables $x_1, \ldots, x_m$. The* influence *of a set $S \subseteq \{x_1, \ldots, x_m\}$ on f, denoted by $I_S(f)$, is defined as the probability that f is undetermined after fixing the variables outside S uniformly at random. Let $I_q(f) = \min_{S \subseteq \{x_1,\ldots,x_m\}, |S| \le q} I_S(f)$. We say that f is* (q, ε)-resilient *if $I_q(f) \le \varepsilon$. In addition, for $0 < \tau < 1$, we say f is* τ-strongly resilient *if for all $1 \le q \le n$, $I_q(f) \le \tau \cdot q$.*

We use the $(\Theta(\log^2 m/m))$-*strongly resilient* function defined in [55] (i.e., any coalition of q bits has influence at most $\Theta(q \cdot \log^2 m/m)$) which has a bias $1/2 \pm 1/10$. We note that it has been shown that for any Boolean function on $m^{O(1)}$ bits, even one bit can have influence $\Omega(\log m/m^{O(1)})$ [39]. Hence, it is not possible to get a single bit string with $\varepsilon = m^{-\Omega(1)}$. In Supplementary Material A.2 we provide further discussion about related work.

Publicly samplable adaptive one-way functions. Adaptive one-way functions (adaptive OWFs, or AOWFs for short) were formally introduced by Pandey *et al.* [58]. In a nutshell, a family of AOWFs is indexed by a tag, $\mathsf{tag} \in \{0,1\}^\lambda$, such that for any tag, it is hard for any PPT adversary to invert $f_{\mathsf{tag}}(\cdot)$ for randomly sampled images, even when given access to the *inversion oracle* of $f_{\mathsf{tag}'}(\cdot)$ for any other $\mathsf{tag}' \neq \mathsf{tag}$. We recall the definition of AOWFs in Supplementary Material A.3. Here, we define a variant of AOWFs where the adversary is provided a *publicly sampled* image as inversion challenge.

Definition 2. *Let $\mathbf{F} = \left\{\{f_{\mathsf{tag}} : X_{\mathsf{tag}} \longrightarrow Y_{\mathsf{tag}}\}_{\mathsf{tag} \in \{0,1\}^\lambda}\right\}_{\lambda \in \mathbb{N}}$ be an AOWF family. We say that $\mathbf{F}$ is* publicly samplable adaptive one-way (PS-AOWF) *if:*

***(1)** There is an efficient deterministic image-mapping algorithm $\mathsf{IM}(\cdot,\cdot)$ such that for every $\mathsf{tag} \in \{0,1\}^\lambda$, it holds that*

$$\Pr\left[\omega \leftarrow \mathbb{U}_\lambda : \mathsf{IM}(\mathsf{tag}, \omega) \in Y_{\mathsf{tag}}\right] = 1 - \mathsf{negl}(\lambda) \ .$$

***(2)** Let $\mathcal{O}(\mathsf{tag}, \cdot, \cdot)$ denote the* inversion oracle *(as in [58]) that, on input tag' and y outputs $f^{-1}_{\mathsf{tag}'}(y)$ if $\mathsf{tag}' \neq \mathsf{tag}$, $|\mathsf{tag}'| = |\mathsf{tag}|$, and $\perp$ otherwise. Then, for every PPT adversary $\mathcal{A}$ and every $\mathsf{tag} \in \{0,1\}^\lambda$, it holds that*

$$\Pr\left[\omega \leftarrow \mathbb{U}_\lambda : \mathcal{A}^{\mathcal{O}(\mathsf{tag},\cdot,\cdot)}(\mathsf{tag}, \omega) = f^{-1}_{\mathsf{tag}}(\mathsf{IM}(\mathsf{tag}, \omega))\right] = \mathsf{negl}(\lambda) \ .$$

For notation simplicity, in the rest of the paper we omit indexing by $\lambda \in \mathbb{N}$ and simply write $\mathbf{F} = \{f_{\mathsf{tag}} : X_{\mathsf{tag}} \longrightarrow Y_{\mathsf{tag}}\}_{\mathsf{tag} \in \{0,1\}^\lambda}$.

The main difference between PS-AOWFs and AOWFs as used in [58] is *public samplability*: even if $\mathcal{A}$ is given the random coins, ω, used for the image mapping algorithm $\mathsf{IM}(\cdot,\cdot)$, it can only invert the OWF with negligible probability. In Supplementary Material A.4, we provide an instantiation of a PS-OWF based on the hardness of discrete logarithm problem (DLP) in the generic group model.

Externalized UC with global helper. Universal Composability (UC) is a widely accepted simulation-based model to analyze protocol security. In the UC framework, all the ideal functionalities are “subroutine respectful” in the sense that each protocol execution session has its own copy of the functionalities, which only interact with the single protocol session. This subroutine respecting feature does not always naturally reflect the real world scenarios; for instance, we typically want the trusted setup (e.g., CRS or PKI) to be deployed once and then used in multiple protocols. To handle global setups the generalized UC (GUC) framework was introduced [17]. However, as noted in the introduction, given that in this work we want to avoid the use of a trusted setup (beyond a consistent bulletin board), while still providing a composable construction, we revert to the extended UC model with super-polynomial time helpers, denoted by $\mathcal{H}$-EUC [18]. In this model both the simulator and the adversary can access a (externalized super-polynomial time) global *helper* functionality $\mathcal{H}$.

3 CVZK and Coalescence Functions

A *crowd verifiable zero-knowledge* (CVZK) argument for a language $\mathcal{L} \in \mathbf{NP}$ with a witness relation $R_{\mathcal{L}}$ is an interactive proof between a PPT prover, that consists of a pair of algorithms $\mathsf{CVZK}.P = (\mathsf{CVZK.Prv}_1, \mathsf{CVZK.Prv}_2)$, and a *collection of PPT verifiers* $(\mathsf{CVZK}.V_1, \ \ldots, \mathsf{CVZK}.V_n)$. The private input of the prover is some witness w s.t. $(x, w) \in R_{\mathcal{L}}$, where x is a public statement. In a CVZK argument execution, the interaction is in three moves as follows:
(1) The prover $\mathsf{CVZK.Prv}_1(x, w)$ outputs the statement x and a string a to all n verifiers and outputs a state st_P.
(2) For $\ell \in [n]$, each verifier $\mathsf{CVZK}.V_\ell(x, a)$ sends a challenge c_ℓ to the prover and keeps a private state st_ℓ (e.g., the coins of V_ℓ). Note that $\mathsf{CVZK}.V_\ell$ gets as input only (x, a), and computes her challenge independently from the other verifiers.
(3) After receiving c_ℓ for all $\ell = \{1, \ldots, n\}$, $\mathsf{CVZK.Prv}_2$ outputs its response, z.

Additionally, there is a verification algorithm $\mathsf{CVZK.Verify}$ that takes as input the execution transcript $\langle x, a, \langle c_\ell \rangle_{\ell \in [n]}, z \rangle$ and optionally, a state st_ℓ, $\ell \in [n]$ (if run by $\mathsf{CVZK}.V_\ell$), and outputs $0/1$.

As discussed in the introduction, CVZK is particularly interesting when each verifier contributes limited (human-level) randomness individually, yet the randomness of all verifiers (seen as a crowd) provides enough entropy to support the protocol's soundness. This unique feature of CVZK will be in the core of the security analysis of our VMPC construction (Sec. 7). Nonetheless, from a mere definitional aspect, the verifiers need not to be limited, so for generality, we pose no restrictions on the entropy of their individual challenges in our definition.

3.1 CVZK Definition

We consider an adversary that statically corrupts up to a ratio of the verifier crowd. Let $\mathcal{I}_{\mathsf{corr}}$ be the set of indices of corrupted verifiers.

Definition 3. *Let n be a positive integer, $0 \leq t_1, t_2, t_3 \leq n$ be positive values and $\epsilon_1(\cdot), \epsilon_2(\cdot)$ be real functions. A tuple of PPT algorithms* $\langle (\mathsf{CVZK.Prv}_1, \mathsf{CVZK.Prv}_2),$

$(\mathsf{CVZK}.V_1, \ldots, \mathsf{CVZK}.V_n), \mathsf{CVZK.Verify}\rangle$ *is a* $(t_1, t_2, t_3, \epsilon_1, \epsilon_2)$-crowd-verifiable zero-knowledge argument of membership (CVZK-AoM) *for a language* $\mathcal{L} \in \mathbf{NP}$*, if the following properties are satisfied:*

(i). (t_1, ϵ_1)***-Crowd-Verifiable Completeness:*** *For every* $x \in \mathcal{L} \cap \{0,1\}^{\mathrm{poly}(\lambda)}$*,* $w \in R_{\mathcal{L}}(x)$*, every PPT adversary* $\mathcal{A}$ *and every* $\mathcal{I}_{\mathsf{corr}} \subseteq [n]$ *such that* $|\mathcal{I}_{\mathsf{corr}}| \le t_1$*, the probability that the following experiment returns 1 is less or equal to* $\epsilon_1(\lambda)$*.*

$\mathbf{Expt}^{\mathsf{CVCompl}}_{(t_1, \mathcal{A}, \mathcal{I}_{\mathsf{corr}})}(1^\lambda, x, w)$

```
1. $\mathsf{CVZK.Prv}_1(x, w)$ outputs the message $a$ and state $\mathsf{st}_P$;
2. For $\ell \in [n] \setminus \mathcal{I}_{\mathsf{corr}}$, run $\mathsf{CVZK}.V_\ell(x, a) \to (c_\ell, \mathsf{st}_\ell)$;
3. $\mathcal{A}\big(x, a, \langle c_\ell \rangle_{\ell \in [n] \setminus \mathcal{I}_{\mathsf{corr}}}\big)$ outputs $\langle c'_1, \ldots, c'_n \rangle$;
4. $\mathsf{CVZK.Prv}_2(x, w, a, \langle c'_1, \ldots, c'_n \rangle, \mathsf{st}_P)$ outputs response $z$;
5. If $(\forall \ell \in [n] \setminus \mathcal{I}_{\mathsf{corr}} : c'_\ell = c_\ell)$ AND $\big((\mathsf{CVZK.Verify}(x, a, \langle c'_1, \ldots, c'_n \rangle, z) = 0)$ OR
   $(\exists \ell \in [n] \setminus \mathcal{I}_{\mathsf{corr}} : \mathsf{CVZK.Verify}(x, a, \langle c'_1, \ldots, c'_n \rangle, z, \mathsf{st}_\ell) = 0)\big)$ then return 1; else return $0$;
```

(ii). (t_2, ϵ_2)***-Crowd-Verifiable Soundness:*** *For every* $x \in \{0,1\}^{\mathrm{poly}(\lambda)} \setminus \mathcal{L}$*, every PPT adversary* $\mathcal{A}$ *and every* $\mathcal{I}_{\mathsf{corr}} \subseteq [n]$ *such that* $|\mathcal{I}_{\mathsf{corr}}| \le t_2$*, the probability that the following experiment returns 1 is less or equal to* $\epsilon_2(\lambda)$*.*

$\mathbf{Expt}^{\mathsf{CVSound}}_{(t_2, \mathcal{A}, \mathcal{I}_{\mathsf{corr}})}(1^\lambda, x)$

```
1. $\mathcal{A}(x, \mathcal{I}_{\mathsf{corr}})$ outputs a message $a$;
2. For $\ell \in [n] \setminus \mathcal{I}_{\mathsf{corr}}$, run $\mathsf{CVZK}.V_\ell(x, a) \to (c_\ell, \mathsf{st}_\ell)$;
3. $\mathcal{A}\big(x, a, \langle c_\ell \rangle_{\ell \in [n] \setminus \mathcal{I}_{\mathsf{corr}}}\big)$ outputs $\langle c'_1, \ldots, c'_n \rangle$ and response $z$;
4. If $(\forall \ell \in [n] \setminus \mathcal{I}_{\mathsf{corr}} : c'_\ell = c_\ell)$ AND $(\mathsf{CVZK.Verify}(x, a, \langle c'_1, \ldots, c'_n \rangle, z) = 1)$ AND
   $(\forall \ell \in [n] \setminus \mathcal{I}_{\mathsf{corr}} : \mathsf{CVZK.Verify}(x, a, \langle c'_1, \ldots, c'_n \rangle, z, \mathsf{st}_\ell) = 1)$
   then return 1 else return $0$;
```

(iii). t_3***-Crowd-Verifiable Zero-Knowledge:*** *For every* $x \in \mathcal{L} \cap \{0,1\}^{\mathrm{poly}(\lambda)}$*,* $w \in R_{\mathcal{L}}(x)$*, every PPT adversary* $\mathcal{A}$ *and every* $\mathcal{I}_{\mathsf{corr}} \subseteq [n]$ *such that* $|\mathcal{I}_{\mathsf{corr}}| \le t_3$*, there is a PPT simulator* $\mathsf{CVZK.Sim} = (\mathsf{CVZK.Sim}_1, \mathsf{CVZK.Sim}_2)$ *such that the outputs of the following two experiments are computationally indistinguishable.*

$\mathbf{Expt}^{\mathsf{CVZK}}_{(\mathsf{Ideal}, t_3, \mathcal{A}, \mathcal{I}_{\mathsf{corr}})}(1^\lambda, x)$

```
1. $\mathsf{CVZK.Sim}_1(x, \mathcal{I}_{\mathsf{corr}})$ outputs $a$, $\mathsf{st}_{\mathsf{Sim}}$,
   and $\langle c_\ell \rangle_{\ell \in [n] \setminus \mathcal{I}_{\mathsf{corr}}}$;
2. $\mathcal{A}\big(x, a, \langle c_\ell \rangle_{\ell \in [n] \setminus \mathcal{I}_{\mathsf{corr}}}\big)$ outputs $\langle c'_1, \ldots, c'_n \rangle$;
3. $\mathsf{CVZK.Sim}_2(x, a, \langle c'_1, \ldots, c'_n \rangle, \mathsf{st}_{\mathsf{Sim}})$
   outputs $z$;
4. $b \leftarrow \mathcal{A}(x, z)$;
5. If $(\forall \ell \in [n] \setminus \mathcal{I}_{\mathsf{corr}} : c'_\ell = c_\ell)$,
   then return $b$; else return $\perp$ ;
```

$\mathbf{Expt}^{\mathsf{CVZK}}_{(\mathsf{Real}, t_3, \mathcal{A}, \mathcal{I}_{\mathsf{corr}})}(1^\lambda, x, w)$

```
1. $\mathsf{CVZK.Prv}_1(x, w)$ outputs $a$ and state $\mathsf{st}_P$;
2. For $\ell \in [n] \setminus \mathcal{I}_{\mathsf{corr}}$, run
   $\mathsf{CVZK}.V_\ell(x, a) \to (c_\ell, \mathsf{st}_\ell)$;
3. $\mathcal{A}\big(x, a, \langle c_\ell \rangle_{\ell \in [n] \setminus \mathcal{I}_{\mathsf{corr}}}\big)$ outputs $\langle c'_1, \ldots, c'_n \rangle$;
4. $\mathsf{CVZK.Prv}_2(x, w, a, \langle c'_1, \ldots, c'_n \rangle, \mathsf{st}_P)$ outputs $z$;
5. $b \leftarrow \mathcal{A}(x, z)$;
6. If $(\forall \ell \in [n] \setminus \mathcal{I}_{\mathsf{corr}} : c'_\ell = c_\ell)$,
   then return $b$; else return $\perp$ ;
```

Analogously, we can also define a CVZK argument of knowledge as follows. We say that $\langle(\mathsf{CVZK.Prv}_1, \mathsf{CVZK.Prv}_2), (\mathsf{CVZK}.V_1, \ldots, \mathsf{CVZK}.V_n), \mathsf{CVZK.Verify}\rangle$ *is* a $(t_1, t_2, t_3, \epsilon_1)$-crowd-verifiable zero-knowledge argument of knowledge (CVZK-AoK)*, if it satisfies* (t_1, ϵ_1)*-Completeness and* t_3*-Crowd-Verifiable Zero-Knowledge as previously, and the following property:*

t_2***- Crowd-Verifiable Validity:*** *There exists a PPT extractor* $\mathsf{CVZK.Ext}$ *such that for every* $x \in \{0,1\}^{\mathrm{poly}(\lambda)}$*, every PPT adversary* $\mathcal{A}$ *and every* $\mathcal{I}_{\mathsf{corr}} \subseteq [n]$

such that $|\mathcal{I}_{\mathsf{corr}}| \leq t_2$, the following holds: if there is a non-negligible function $\alpha(\cdot)$ such that

$$\Pr\left[\mathbf{Expt}^{\mathsf{CVSound}}_{(t_2,\mathcal{A},\mathcal{I}_{\mathsf{corr}})}(1^\lambda, x) = 1\right] \geq \alpha(\lambda)\ ,$$

then there is a non-negligible function $\beta(\cdot)$ such that

$$\Pr[w^* \leftarrow \mathsf{CVZK.Ext}^{\mathcal{A}}(x, \mathcal{I}_{\mathsf{corr}}) : (x, w^*) \in R_{\mathcal{L}}] \geq \beta(\lambda)\ .$$

Remark 1 (Relativized CVZK security). Definition 3 specifies CVZK security against a PPT adversary $\mathcal{A}$ and a PPT simulator $\mathsf{CVZK.Sim}$. Note that the notions of crowd-verifiable completeness, soundness, validity, and zero-knowledge can be extended so that they hold even when $\mathcal{A}$, and maybe $\mathsf{CVZK.Sim}$, has also access to a (potentially super-polynomial) oracle $\mathcal{H}$.

Remark 2 (Sensitivity of CVZK soundness to the number of verifiers). In our definition of CVZK soundness, even a single (out of t_2) honest failed verification invalidates a soundness attack. We could extend the definition of soundness by adding a parameter s that captures tolerance of failed verifications. However, given the independency of verifications in our setting (every verifier decides based on her own private state), this extension would not be much more expressive security-wise in practice. We detail our argumentation in Supplementary Material B.

3.2 Coalescence Functions

We introduce the notion of a *coalescence function*, which will be a core component of our CVZK construction (cf. Section 4). In particular, coalescence functions will be the key for exploiting the CVZK verifiers' randomness in the presence of an adversary (a malicious prover) that aims to "cancel" the entropy of the honest verifiers. Given the verifiers' coins, a coalescence function will produce a collection of (challenge) strings such that at least one of the strings has sufficient entropy to support CVZK soundness. At a high level, a function F achieves coalescence, if when provided as input an n-dimensional vector that is (i) sampled from a distribution $\mathbb{D}_\lambda$, and (ii) adversarially tampered at up to t-out-of-n vector components, it outputs a sequence of m k-bit strings so that with overwhelming probability, at least one of the m strings is statistically close to uniformly random. Our definition of F postulates the existence of "good" events $\mathsf{G}_1, \ldots \mathsf{G}_m$, defined over the input distribution, where conditional to G_i being true, the corresponding output string is statistically close to uniform. Coalescence is achieved if the probability that such a "good" event occurs is overwhelming.

Definition 4. *Let n, k, m be polynomial in λ and $\mathbf{In} = (in^{(1)}, \ldots, in^{(n)})$ be an n-dimensional vector sampled according to the distribution ensemble $\{\mathbb{D}_\lambda\}_\lambda$ so that the support of $\mathbb{D}_\lambda$ is Ω_λ. Let $F : \Omega_\lambda \longrightarrow (\{0,1\}^k)^m$ be a function. For any adversary $\mathcal{A}$, any $t \leq n$, and any $\mathcal{I}_{\mathsf{corr}} \subseteq [n]$ such that $|\mathcal{I}_{\mathsf{corr}}| \leq t$, we define the following experiment:*

$\mathbf{Expt}^{\mathsf{Coal}}_{(t,\mathcal{A},\mathcal{I}_{\mathsf{corr}})}(1^\lambda)$

1. *Set* $\mathbf{In} = (in^{(1)}, \ldots, in^{(n)}) \leftarrow \mathbb{D}_\lambda$*;*
2. $\mathcal{A}\big(\langle in^{(\ell)}\rangle_{\ell\in\mathcal{I}_{\mathsf{corr}}}\big)$ *outputs* $\mathbf{In}' = (in'^{(1)}, \ldots, in'^{(n)})$ *s.t.* $\forall \ell \in [n] \setminus \mathcal{I}_{\mathsf{corr}} : in'^{(\ell)} = in^{(\ell)}$*;*
3. *Return* $(d_1, \ldots, d_m) \leftarrow F(\mathbf{In}')$*;*

We say that the function $F : \Omega_\lambda \to (\{0,1\}^k)^m$ *is a* (k, m, t)-coalescence function w.r.t. $\mathbb{D}_\lambda$*, if there exist events* $\mathsf{G}_1, \ldots \mathsf{G}_m$ *over* Ω_λ *such that the following two conditions hold:*

(1) $\Pr[\wedge_{i=1}^m \neg \mathsf{G}_i] = \mathsf{negl}(\lambda)$*, and*

(2) *for every adversary* $\mathcal{A}$ *and every* $\mathcal{I}_{\mathsf{corr}} \subseteq [n]$ *such that* $|\mathcal{I}_{\mathsf{corr}}| \leq t$*, it holds that for all* $i \in [m]$*, the random variable* $(d_i|\mathsf{G}_i)$ *is statistically* $\mathsf{negl}(\lambda)$*-close to* $\mathbb{U}_k$*, where* $(d_1, \ldots, d_m) \leftarrow \mathbf{Expt}^{\mathsf{Coal}}_{(t,\mathcal{A},\mathcal{I}_{\mathsf{corr}})}(1^\lambda)$*. Note that* $(X|\mathsf{A})$ *denotes the random variable* X *conditional on the event* A*.*

Furthermore, we require that a (k, m, t)*-coalescence function* F *w.r.t.* $\mathbb{D}_\lambda$ *satisfies the following two additional properties:*

Completeness: *the output of* F *on inputs sampled from* $\mathbb{D}_\lambda$*, denoted by* $F(\mathbb{D}_\lambda)$*, is statistically* $\mathsf{negl}(\lambda)$*-close to the uniform distribution* $(\mathbb{U}_k)^m$ *over* $(\{0,1\}^k)^m$*.*

Efficient samplability: *there exists a PPT algorithm* $\mathsf{Sample}(\cdot)$ *such that the following two conditions hold:*

(a) $\Pr_{(d_1,\ldots,d_m)\leftarrow(\mathbb{U}_k)^m}\big[\mathbf{In} \leftarrow \mathsf{Sample}(d_1, \ldots, d_m) : F(\mathbf{In}) = (d_1, \ldots, d_m)\big] = 1 - \mathsf{negl}(\lambda)$*.*

(b) The distribution $\mathsf{Sample}\big((\mathbb{U}_k)^m\big)$ *is statistically* $\mathsf{negl}(\lambda)$*-close to* $\mathbb{D}_\lambda$*.*

In Section 4.1, we present an implementation of a coalescence function w.r.t. $\mathbb{U}_n$ based on 1RCCF.

4 CVZK Construction

In this section, we show how to compile any Σ-protocol into a 3-move CVZK protocol. Our CVZK construction is a compiler that utilizes an explicit instantiation of a coalescence function from 1RCCF and a special class of protocols where both the prover and the simulator operate in an “input-delayed” manner, i.e., they do not need to know the statement in the first move. Our CVZK protocol will be a basic tool for the construction of our VMPC scheme (cf. Section 7). As noted in the introduction, the security of the VMPC scheme is in the extended UC model (EUC), where both the simulator and the adversary have access to a (externalized super-polynomial time) global helper functionality $\mathcal{H}$, denoted as $\mathcal{H}$-EUC security. Therefore, the CVZK protocol must also be secure against PPT adversaries with oracle access to some helper.

4.1 Coalescence Functions from 1RCCF

As mentioned in Section 2, it is not possible to produce a *single* random string via collective coin flipping and hope it has exponentially small statistical distance to a uniformly random string. Nevertheless, we show that it is possible to produce *several* random strings such that with overwhelming probability *one of them* is close to uniformly random, as dictated by the coalescence property.

Description. Without loss of generality, let $n = \lambda^\gamma$ for a constant $\gamma > 1$ and assume $\lambda \log \lambda$ divides n; if not, we can always pad with 0's until the total number of coins is a multiple of $\lambda \log \lambda$. Let f_{res} denote the $(\Theta(\log^2 m/m))$-strongly resilient function over m bits proposed in [55]. We define the instantiation of the coalescence function $F : \{0,1\}^n \longrightarrow \left(\{0,1\}^{\frac{\lambda}{\log^2 \lambda}}\right)^{\log \lambda}$ as follows:

Step 1. On input $C := (c_1, \ldots, c_n)$, F partitions the n-bit input C into $\lambda \log \lambda$ blocks $B_1, \ldots, B_{\lambda \log \lambda}$, with $\frac{n}{\lambda \log \lambda}$ bits each. Namely $B_j := \left(c_{\frac{(j-1)n}{\lambda \log \lambda}+1}, \ldots, c_{\frac{jn}{\lambda \log \lambda}}\right)$, where $j \in [\lambda \log \lambda]$.

Step 2. Then, F groups every λ blocks together, resulting to $\log \lambda$ groups, denoted as $G_1, \ldots, G_{\log \lambda}$. Namely, $G_i := \left(B_{(i-1)\lambda+1}, \ldots, B_{i\lambda}\right)$, where $i \in [\log \lambda]$. Within each group G_i, we apply the resilient function f_{res} on each block $B_{(i-1)\lambda+k}$, $k \in [\lambda]$, to output 1 bit; hence, for each group G_i, by sequentially running f_{res} we obtain a λ-bit string $(b_{i,1}, \ldots, b_{i,\lambda}) \leftarrow \left(f_{\mathsf{res}}(B_{(i-1)\lambda+1}), \ldots, f_{\mathsf{res}}(B_{i\lambda})\right)$, and $\log \lambda$ strings in total for all the groups G_i, $i \in [\log \lambda]$.

Step 3. The resilient function f_{res} in [55] has a bias $\frac{1}{10}$. Therefore, even if the input G_i is random, the output bits $(b_{i,1}, \ldots, b_{i,\lambda})$ are not a random sequence of $\lambda \log \lambda$ bits due to this bias. In order to make the output of F balanced (i.e., unbiased), for each group G_i, $i \in [\log \lambda]$, we execute the following process: on input $(b_{i,1}, \ldots, b_{i,\lambda})$, we perform a *sequential (von Neumann) rejection sampling* over pairs of bits until an unbiased string $d_i := (d_{i,1}, \ldots, d_{i,\frac{\lambda}{\log^2 \lambda}})$ is produced, with $\frac{\lambda}{\log^2 \lambda}$ bits length as described below:

```
1. Set two indices $j \leftarrow 1$ and $k \leftarrow 1$;
2. **While** $\left((j < \lambda) \wedge (k < \frac{\lambda}{\log^2 \lambda})\right)$:
   – **If** $b_{i,j} \neq b_{i,j+1}$, **then** set $d_{i,k} \leftarrow b_{i,j}$ and $k \leftarrow k+1$;
   – Set $j \leftarrow j+2$;
3. **If** $k = \frac{\lambda}{\log^2 \lambda}$, **then** return $d_i := (d_{i,1}, \ldots, d_{i,\frac{\lambda}{\log^2 \lambda}})$;
4. **else** return $d_i := (b_{i,1}, \ldots, b_{i,\frac{\lambda}{\log^2 \lambda}})$;
```

Finally, we define the output of $F(C)$ as the sequence $(d_1, \ldots, d_{\log \lambda})$.

Security. The security of $F(\cdot)$ is stated below and is proved in C.1.

Theorem 1. *Let $\gamma > 1$ be a constant and $n = \lambda^\gamma$. Then, the function $F : \{0,1\}^n \longrightarrow \left(\{0,1\}^{\frac{\lambda}{\log^2 \lambda}}\right)^{\log \lambda}$ described in Section 4.1 is a $\left(\frac{\lambda}{\log^2 \lambda}, \log \lambda, \frac{n^{1-\frac{1}{\gamma}}}{\log^3 n}\right)$-coalescence function w.r.t. uniform distribution $\mathbb{U}_n$ that satisfies completeness and efficient samplability.*

By Theorem 1, for $n = \lambda^\gamma$, if the adversary can corrupt up to $\frac{n^{1-\frac{1}{\gamma}}}{\log^3 n}$ verifiers, then on input the n verifiers' coins, F outputs $\log \lambda$ strings of $\frac{\lambda}{\log^2 \lambda}$ bits, such that with probability $1 - \mathsf{negl}(\lambda)$, at least one of the $\log \lambda$ strings is statistically close to uniformly random.

4.2 A helper family for AOWF inversion

Let $\mathbf{F} = \{f_{\mathsf{tag}} : X_{\mathsf{tag}} \longrightarrow Y_{\mathsf{tag}}\}_{\mathsf{tag}\in\{0,1\}^\lambda}$ be a (publicly samplable) AOWF family. In Fig. 1, we define the associated helper family $\mathbf{H} = \{\mathcal{H}_S\}_{S\subset\{0,1\}^\lambda}$ (we omit indexing by $\lambda \in \mathbb{N}$ for simplicity). Here, S refers to the subset of tags of entities controlled by an adversary. Namely, the adversary can only ask for preimages that are consistent with its corruption extent.

The helper $\mathcal{H}_S(\cdot,\cdot)$, where $S \subset \{0,1\}^\lambda$.

On query (tag, β), if $\mathsf{tag} \in S$, then it returns a value $\alpha \in X_{\mathsf{tag}}$ s.t. $f_{\mathsf{tag}}(\alpha) = \beta$. Otherwise, it returns $\perp$.

Fig. 1: The helper family $\mathbf{H} = \{\mathcal{H}_S\}_{S\subset\{0,1\}^\lambda}$ w.r.t. $\mathbf{F} = \{f_{\mathsf{tag}}\}_{\mathsf{tag}\in\{0,1\}^\lambda}$.

4.3 Fully Input-delayed Σ-protocols

In our CVZK construction, we utilize a special class of Σ-protocols where both the prover and the simulator do not need to know the proof statement in the first move. Such "input-delayed" protocols (at least for the prover side) have been studied in the literature (e.g., [49, 21, 22, 37]). To stress the input-delayed property for both prover and simulator, we name these protocols *fully input-delayed* and provide their definition below.

Definition 5. *Let $\Sigma.\Pi := (\Sigma.\mathsf{Prv}_1, \Sigma.\mathsf{Prv}_2, \Sigma.\mathsf{Verify})$ be a Σ-protocol for a language $\mathcal{L} \in$* **NP**. *We say that $\Sigma.\Pi$ is* fully input-delayed *if for every $x \in \mathcal{L}$, it satisfies the following two properties:*
(1) Input-delayed proving*: $\Sigma.\mathsf{Prv}_1$ takes as input only the length of x, $|x|$.*
(2) Input-delayed simulation*: there exists an sHVZK simulator $\Sigma.\mathsf{Sim} := (\Sigma.\mathsf{Sim}_1, \Sigma.\mathsf{Sim}_2)$ s.t. $\Sigma.\mathsf{Sim}_1$ takes as input only $|x|$ and the challenge c.*

As we will see in Section 4.4, CVZK can be built upon any fully input-delayed protocol (in a black-box manner) for a suitable "one-way" language that is secure against helper-aided PPT adversaries. Here, for generality, we propose an instantiation of such a protocol from the fully input-delayed proof for the Hamiltonian Cycle problem of Lapidot and Shamir (LS) [49]. By the LS protocol, we know that there exists a fully input-delayed Σ-protocol for every **NP** language. In C.3, we recall the LS protocol and show that it is secure against helper-aided PPT adversaries, when built upon a commitment scheme that is also secure against PPT adversaries with access to the same helper. In C.2, we propose an instantiation of such a commitment scheme based on ElGamal, assuming an "adaptive" variant of the DDH problem in the spirit of AOWFs [58], defined formally in A.4, Definition 10.

4.4 Generic CVZK Compiler

We present a generic CVZK compiler for any Σ-protocol $\Sigma.\Pi = (\Sigma.\mathsf{Prv}_1, \Sigma.\mathsf{Prv}_2, \Sigma.\mathsf{Verify})$ for an **NP** language $\mathcal{L}$ and $(x, w) \in \mathcal{R}_{\mathcal{L}}$. Let $\mathbf{F} = \{f_{\mathsf{tag}} :$

$X_{\mathsf{tag}} \longrightarrow Y_{\mathsf{tag}}\}_{\mathsf{tag}\in\{0,1\}^{\lambda/\log^2\lambda}}$ be a PS-AOWF family (cf. Definition 2), and tag_ℓ be the identity of the verifier $\mathsf{CVZK}.V_\ell$ for $\ell \in [n]$. Let $|\mathsf{tag}_1| = \cdots = |\mathsf{tag}_n|$. For each $\ell \in [n]$, our compiler utilizes a fully input-delayed Σ-protocol $\mathsf{InD}.\Pi := (\mathsf{InD.Prv}_1, \mathsf{InD.Prv}_2, \mathsf{InD.Verify})$ for the language $\mathcal{L}^*_{\mathsf{tag}_\ell}$ defined as:

$$\mathcal{L}^*_{\mathsf{tag}_\ell} = \left\{\beta \in Y_{\mathsf{tag}_\ell} \;\middle|\; \exists \alpha \in X_{\mathsf{tag}_\ell} : f_{\mathsf{tag}_\ell}(\alpha) = \beta\right\} . \tag{1}$$

For simplicity, we say that $\mathsf{InD}.\Pi$ is for the family $\left\{\mathcal{L}^*_{\mathsf{tag}_\ell}\right\}_{\ell\in[n]}$, without referring specifically to the family member.

Description. In terms of architecture, our CVZK compiler is in the spirit of disjunctive proofs [24, 22]: the prover must show that either (i) *knows a witness* w for $x \in \mathcal{L}$ or (ii) *can invert a hard instance* of the PS-AOWF f_{tag}. However, several adaptations are required so that validity and ZK are preserved in the CVZK setting where multiple (individually weak) verifiers are present. First, the challenge C provided by the n verifiers is given as input to the coalescence function $F(\cdot)$ defined in Sec. 4.1 which outputs $\log\lambda$ strings $(d_1, \ldots, d_{\mathsf{log}\lambda})$, each $\frac{\lambda}{\log^2\lambda}$ bits long. In addition, the compiler maintains a fixed disjunctive mode so that the prover always (i) proves the knowledge of w for $x \in \mathcal{L}$ and (ii) simulates the knowledge of a collection of inversions to hard instances.

To prove the knowledge of w for $x \in \mathcal{L}$, the prover executes $\log\lambda$ parallel runs of the compiled Σ-protocol $\Sigma.\Pi$ for $(x, w) \in \mathcal{R}_{\mathcal{L}}$, where the challenge in the i-th run is the XOR operation of the i-th block of $\frac{n}{\log\lambda}$ verifiers' bits from C and some randomness provided by the prover in the first move. To simulate the inversions to hard instances, our compiler exploits the fully input-delayed property of $\mathsf{InD}.\Pi$. In particular, it runs $n \cdot \log\lambda$ parallel simulations of $\mathsf{InD}.\Pi$ where the (ℓ, j)-th run, $(\ell, j) \in [n] \times [\log\lambda]$, is for a hard instance (statement) $x^*_{\ell,j}$ associated with the identity tag_ℓ of $\mathsf{CVZK}.V_\ell$. The statement $x^*_{\ell,j}$ is created later on in the third move of the protocol by running the image-mapping algorithm of $\mathbf{F}$ on input tag_ℓ and the j-th string output by $F(C)$, d_j. The latter is feasible because the first move of the input-delayed simulator $\mathsf{InD.Sim}$ is executed obliviously to the statement.

By the coalescence property of $F(\cdot)$, the output $F(C)$ preserves enough entropy, so that any malicious CVZK prover corrupting less than $\frac{n^{1-\frac{1}{\gamma}}}{\log^3 n}$ verifiers is forced to be challenged on the knowledge of (i) w for $x \in \mathcal{L}$ or (ii) an inversion of a hard instance, in at least one of the corresponding parallel executions. Thus, by the adaptive one-way property of $\mathbf{F}$, the (potentially malicious) prover must simulate the knowledge of all inversions and *indeed prove* the knowledge of w for $x \in \mathcal{L}$, so CVZK validity is guaranteed.

The ZK property of our compiler relies on the sHVZK properties of $\Sigma.\Pi$ and $\mathsf{InD}.\Pi$, yet we remark that the CVZK simulation must be *straight-line* (no rewindings) so that our construction can be deployed in the $\mathcal{H}$-EUC setting of our VMPC scheme. For this reason, we do "complexity leveraging" along the lines of super-polynomial simulation introduced in [59], by allowing our simulator to have access to members of the helper family $\mathbf{H}$ defined in Fig. 1. Our CVZK compiler is presented in detail in Fig. 2.

1. $\mathsf{CVZK.Prv}_1(x, w)$:
   - For $i \in [\log \lambda]$, run $(a_i, \mathsf{st}_i) \leftarrow \Sigma.\mathsf{Prv}_1(x, w)$.
   - Pick random $R := (r_1, \ldots, r_n) \leftarrow \{0, 1\}^n$.
   - For $\ell \in [n]$ and $j \in [\log \lambda]$, run $(a^*_{\ell,j}, \mathsf{st}^*_{\ell,j}) \leftarrow \mathsf{InD.Sim}_1(r_\ell, size)$, where $size = \log \lambda \cdot |M_{\frac{\lambda}{\log^2 \lambda}}(\mathsf{tag}_\ell, \cdot)|$ and $|M_{\frac{\lambda}{\log^2 \lambda}}(\mathsf{tag}_\ell, \cdot)|$ is the circuit size of $f_{\mathsf{tag}_\ell}(\cdot)$ as in Definition 2.
   - Output $A := \left(\{a_i\}_{i \in [\log \lambda]}, \{a^*_{\ell,j}\}^{j \in [\log \lambda]}_{\ell \in [n]}\right)$ and the state $\mathsf{st}_P := \left(R, \{\mathsf{st}_i\}_{i \in [\log \lambda]}, \{\mathsf{st}^*_{\ell,j}\}^{j \in [\log \lambda]}_{\ell \in [n]}\right)$.
2. The verifiers generate coins $C := (c_1, \ldots, c_n) \in \{0, 1\}^n$. I.e., for $\ell \in [n]$, $\mathsf{CVZK}.V_\ell(x, a)$ outputs a random bit c_ℓ.
3. $\mathsf{CVZK.Prv}_2(x, w, A, C, \mathsf{st}_P)$:
   - Parse $\mathsf{st}_P := \left(R, \{\mathsf{st}_i\}_{i \in [\log \lambda]}, \{\mathsf{st}^*_{\ell,j}\}^{j \in [\log \lambda]}_{\ell \in [n]}\right)$.
   - Compute the coalescence function $F(\cdot)$ defined in Section 4.1 on input C to get $F(C) = (d_1, \ldots, d_{\log \lambda})$, where $d_j \in \{0,1\}^{\lambda/\log^2 \lambda}$, $j \in [\log \lambda]$.
   - Set $E := R \oplus C$, and parse E as $(e_1, \ldots, e_{\log \lambda})$, where $e_i \in \{0,1\}^{n/\log \lambda}$.
   - For $i \in [\log \lambda]$, run $z_i \leftarrow \Sigma.\mathsf{Prv}_2(\mathsf{st}_i, e_i)$.
   - For $\ell \in [n]$ and $j \in [\log \lambda]$:
     - Run $\beta_{\ell,j} \leftarrow \mathsf{IM}(\mathsf{tag}_\ell, d_j)$, where $\mathsf{IM}(\cdot, \cdot)$ is the image-mapping algorithm of the family $\mathbf{F}$, as in Definition 2.
     - Define the statement $x^*_{\ell,j} := \beta_{\ell,j}$ for $\mathcal{L}^*_{\mathsf{tag}_\ell}$.
     - Run $z^*_{\ell,j} \leftarrow \mathsf{InD.Sim}_2(\mathsf{st}^*_{\ell,j}, x^*_{\ell,j})$.
   - Output $Z := \left(E, \{z_i\}_{i \in [\log \lambda]}, \{z^*_{\ell,j}\}^{j \in [\log \lambda]}_{\ell \in [n]}\right)$.
4. $\mathsf{CVZK.Verify}(x, A, C, Z)$:
   - Parse $A := \left(\{a_i\}_{i \in [\log \lambda]}, \{a^*_{\ell,j}\}^{j \in [\log \lambda]}_{\ell \in [n]}\right)$.
   - Parse $Z := \left(E, \{z_i\}_{i \in [\log \lambda]}, \{z^*_{\ell,j}\}^{j \in [\log \lambda]}_{\ell \in [n]}\right)$.
   - Compute $(d_1, \ldots, d_{\log \lambda}) \leftarrow F(C)$.
   - Compute $R := (r_1, \ldots, r_n) = E \oplus C$ and parse E as $(e_1, \ldots, e_{\log \lambda})$.
   - For $i \in [\log \lambda]$, check that $\Sigma.\mathsf{Verify}(x, a_i, e_i, z_i) = 1$.
   - For $\ell \in [n]$ and $j \in [\log \lambda]$, run $\beta_{\ell,j} \leftarrow \mathsf{IM}(\mathsf{tag}_\ell, d_j)$ and define the statement $x^*_{\ell,j} = \beta_{\ell,j}$. Then, check that $\mathsf{InD.Verify}\left(x^*_{\ell,j}, a^*_{\ell,j}, r_\ell, z^*_{\ell,j}\right) = 1$.
   - Output 1 if all the checks are valid; output 0, otherwise.

Fig. 2: The generic CVZK compiler $\mathsf{CVZK}.\Pi$.

Security. To prove the security of our CVZK generic compiler we use a simulator pair $(\mathsf{CVZK.Sim}_1, \mathsf{CVZK.Sim}_2)$, where $\mathsf{CVZK.Sim}_2$ is given oracle access to a member of the super-polynomial helper family $\mathbf{H} = \{\mathcal{H}_S\}_{S \subset \{0,1\}^{\lambda/\log^2 \lambda}}$ defined in Fig. 1. We state our CVZK security theorem below and prove it in Supplementary Material C.4.

Theorem 2. *Let* $\Sigma.\Pi = (\Sigma.\mathsf{Prv}_1, \Sigma.\mathsf{Prv}_2, \Sigma.\mathsf{Verify})$ *be a* Σ*-protocol for some language* $\mathcal{L} \in \mathbf{NP}$ *where the challenge is chosen uniformly at random. Let* $\mathbf{F} = \{f_{\mathsf{tag}} : X_{\mathsf{tag}} \longrightarrow Y_{\mathsf{tag}}\}_{\mathsf{tag} \in \{0,1\}^{\lambda/\log^2 \lambda}}$ *be a PS-AOWF family (cf. Definition 2), and let* $\mathbf{H} = \{\mathcal{H}_S\}_{S \subset \{0,1\}^{\lambda/\log^2 \lambda}}$ *be the associated helper family defined in Fig. 1. Let* $\mathsf{InD}.\Pi := (\mathsf{InD.Prv}_1, \mathsf{InD.Prv}_2, \mathsf{InD.Verify})$ *be a fully input-delayed*

Σ-protocol for the language family $\left\{\mathcal{L}^*_{\mathsf{tag}_\ell}\right\}_{\ell\in[n]}$ defined in Eq.(1).
Let $\gamma > 1$ be a constant and $n = \lambda^\gamma$. Let $\mathsf{CVZK}.\Pi$ be the CVZK compiler for the language $\mathcal{L}$ with n verifiers described in Fig. 2 over $\Sigma.\Pi$, $\mathsf{InD}.\Pi$ and $\mathbf{F}$. Then, against any adversary $\mathcal{A}$, it holds that:
***(1)** If the image-mapping algorithm $\mathsf{IM}(\cdot,\cdot)$ of $\mathbf{F}$ has error $\epsilon(\cdot)$[5], $\Sigma.\Pi$ has completeness error $\delta(\cdot)$ and $\mathsf{InD}.\Pi$ has perfect completeness, then for every $t_1 \leq \frac{n^{1-\frac{1}{\gamma}}}{\log^2 n}$, $\mathsf{CVZK}.\Pi$ satisfies (t_1, ϵ_1)-crowd verifiable completeness, where $\epsilon_1(\lambda) := \delta(\lambda)\log\lambda + n\log\lambda\epsilon(\lambda)2^{\Theta(\log^2 n)} + \mathsf{negl}(\lambda)$.*
***(2)** If $\Sigma.\Pi$ and $\mathsf{InD}.\Pi$ are special sound, then for every $t_2 \leq \frac{n^{1-\frac{1}{\gamma}}}{\log^3 n}$, there is a negligible function $\epsilon_2(\cdot)$ s.t. $\mathsf{CVZK}.\Pi$ satisfies (t_2, ϵ_2)-crowd verifiable soundness and t_2-crowd verifiable validity.*
***(3)** Let $t_3 \leq n$ and consider any subset of indices of corrupted verifiers $\mathcal{I}_{\mathsf{corr}} \subseteq [n]$ s.t. $|\mathcal{I}_{\mathsf{corr}}| \leq t_3$. Let $\mathcal{A}$ be PPT with access to a helper $\mathcal{H}_S$ from $\mathbf{H}$, where (i) $\{\mathsf{tag}_\ell\}_{\ell\in\mathcal{I}_{\mathsf{corr}}} \subseteq S$ and (ii) $\{\mathsf{tag}_\ell\}_{\ell\in[n]\setminus\mathcal{I}_{\mathsf{corr}}} \cap S = \emptyset$. If $\Sigma.\Pi$ and $\mathsf{InD}.\Pi$ are sHVZK against PPT distinguishers with access to $\mathcal{H}_S$, then there is a PPT simulator pair $\left(\mathsf{CVZK}.\mathsf{Sim}_1,\ \mathsf{CVZK}.\mathsf{Sim}_2^{\mathcal{H}_S}\right)$ s.t. $\mathsf{CVZK}.\Pi$ is t_3-crowd-verifiable zero-knowledge against PPT distinguishers with access to $\mathcal{H}_S$.*

5 End-to-end Verifiable MPC

We introduce *end-to-end verifiable multiparty computation (VMPC)*, which as we show in Section 7, can be realized with the use of CVZK. A VMPC scheme encompasses the interaction among sets of *users*, *clients* and *servers*, so that the correct computation of some fixed function f of the users' private inputs can be verified, while their privacy is preserved. End-to-end verifiability suggests that even when *all* servers and *all* users' clients are corrupted, verification is still possible (although, obviously, in an all-malicious setting, privacy is violated). Furthermore, a user's audit data do not leak information about her private input so the verification mechanism may be delegated to an external verifier.

5.1 VMPC Syntax

Let $\mathcal{U} = \{\, U_1, \ldots, U_n \,\}$ be a set of n users where every user has an associated client $\mathcal{C} = \{\, C_1, \ldots, C_n \,\}$. Let $\mathcal{S} = \{\, S_1, \ldots, S_k \,\}$ be a set of k servers. All clients and servers run in polynomial time. Every server has write permission to a consistent *bulletin board* (BB) to which all parties have read access. Each user U_ℓ receives her private input x_ℓ from some set X (which includes a special symbol "$\mathsf{abstain}$") and is associated with a *client* C_ℓ for engaging in the VMPC execution. In addition, there exists an efficient *verifier* V responsible for auditing procedures. The *evaluation function* associated with the VMPC scheme is

[5] The adaptive OWF family instantiated in A.4 has perfect samplability, i.e. $\epsilon(\lambda) = 0$.

denoted by $f : X^n \longrightarrow Y$, where X^n is the set of vectors of length n, the coordinates of which are elements in X, and Y is the range set. All parameters and set sizes n, k are polynomial in the security parameter λ.

Note that we consider the concept of a single verifier that audits the VMPC execution on behalf of the users, in the spirit of delegatable receipt-free verification that is established in e-voting literature (e.g. [20, 56, 44]). Alternatively, we could involve multiple verifiers, e.g. one for each user, and require that all or a threshold of them verify successfully. This approach does not essentially affect the design and security analysis of a VMPC scheme, as (i) individual verifiability is captured in our description via the delegatable verification carried out by the single verifier and (ii) a threshold of collective user randomness is anyway needed. Which of the two directions is preferable, is mostly a matter of deployment and depends on the real world scenario where the VMPC is used.

Separating users from their client devices. The distinction between the user and her associated client is crucial for the analysis of VMPC security where end-to-end verifiability is preserved in an all-malicious setting, i.e., where the honest users are against a severe adversarial environment that controls the entire VMPC execution by corrupting all servers and all clients. In this setting, each user is an entity with limited "human level" power, unable of performing complex cryptographic operations which are outsourced to her associated client. A secure VMPC scheme should be designed in a way that withstands such attacks, based on the engagement of the honest users in the execution.

VMPC security relies on the *internal randomness* that each user generates during her interaction with the system. By r_ℓ we denote the randomness generated by the user U_ℓ and κ_ℓ is the min-entropy of r_ℓ. Let $\kappa := \min\{\kappa_\ell \mid \ell \in [n]\}$ be the min-entropy of all users' randomness, that we call the *user min-entropy* of a VMPC scheme. Given that we view U_ℓ as a "human entity", the values of κ are small and insufficient for secure implementation of cryptographic primitives. Namely, each individual user contributes randomness that can be guessed by an adversary with non-negligible probability. Formally, it should hold $\kappa = O(\mathsf{log}\lambda)$, i.e. $2^{-\kappa}$ is a non-negligible value and hence insufficient for any cryptographic operation. From a computational point of view, users cannot perform complicated calculations and their computational complexity is linear in λ (i.e., the minimum for reading the input).

Protocols. A VMPC scheme consists of the following protocols:

– **Initialize** (executed among the servers). At the end of the protocol each server S_i posts a public value Params_i in the BB and maintains private state st_i. By $\mathsf{Params} = \{\mathsf{Params}_i,\ i \in [k]\}$ we denote the execution's public parameters.

– **Input** (executed among the servers and the users along with their associated clients). We restrict the interaction in the simple setting where the users engage in the **Input** protocol *without interacting* with each other. Specifically, each user U_ℓ, provides her input x_ℓ to her client C_ℓ (e.g., smartphone or desktop PC) which in turn interacts with the servers. By her interaction with C_ℓ, the user U_ℓ obtains some string α_ℓ that will be used as *individual audit data*.

– **Compute** (executed among the servers). At the end of the protocol, the servers post an *output value* y and the *public audit data* τ on the BB. Then, everyone may obtain the output y from the BB.

– **Verify** (executed by the verifier V and the users). In particular, V requests the individual audit data α_ℓ from each user U_ℓ and reads y, τ from the BB. Subsequently it provides each user U_ℓ with a pair (y, v), where $v \in \{0, 1\}$ denotes the verification success or failure.

Remark 3. The **Initialize** protocol can operate as a setup service that is run ahead of time and is used for multiple executions, while the **Input** protocol represents the online interaction between a user, her client and the servers.

5.2 Security Framework

We define a functionality that captures the two fundamental properties that every VMPC should achieve: (i) standard *MPC security* and (ii) *end-to-end verifiability*. Our model for VMPC is in the spirit of $\mathcal{H}$-EUC security [18], which allows for the preservation of the said properties under arbitrary protocol compositions. Thus, VMPC security refers to indistinguishability between an ideal and a real world setting by any environment that schedules the execution. In our definition we assume the functionality of a *Bulletin Board* $\mathcal{G}_{\mathsf{BB}}$ (with consistent write/read operations) and a functionality $\mathcal{F}_{\mathsf{sc}}$ that models a *Secure Channel* between each user and her client (for completeness we recall $\mathcal{G}_{\mathsf{BB}}$ and $\mathcal{F}_{\mathsf{sc}}$ in A.5).

Ideal world setting. We formally describe the *ideal VMPC functionality* $\mathcal{F}^{f,R}_{\mathsf{vmpc}}(\mathcal{P})$ that is defined w.r.t. to an *evaluation function* $f : X^n \longrightarrow Y$ and a *binary relation* $R \subseteq \mathsf{Img[f]} \times \mathsf{Img}[f]$ over the image of f. The functionality $\mathcal{F}^{f,R}_{\mathsf{vmpc}}(\mathcal{P})$ operates with the parties in $\mathcal{P} = \mathcal{U} \cup \mathcal{C} \cup \mathcal{S} \cup \{V\}$, which include the users $\mathcal{U} = \{ U_1, \ldots, U_n \}$ along with their associated clients $\mathcal{C} = \{ C_1, \ldots, C_n \}$, the servers $\mathcal{S} = \{ S_1, \ldots, S_k \}$, and the verifier V.

The relation R determines the level of security offered by $\mathcal{F}^{f,R}_{\mathsf{vmpc}}(\mathcal{P})$ in terms of adversarial manipulation of the output computed value. E.g., if R is the equality relation $\{ (y, y) \mid y \in Y \}$, then no deviation from the actual intended evaluation will be permitted by the $\mathcal{F}^{f,R}_{\mathsf{vmpc}}(\mathcal{P})$. Finally, the environment $\mathcal{Z}$ provides the parties with their inputs and determines a subset $L_{\mathsf{corr}} \subset \mathcal{P}$ of statically corrupted parties. Along the lines of the $\mathcal{H}$-EUC model, we consider an externalized global *helper* functionality $\mathcal{H}$ in both the ideal and real world. The helper $\mathcal{H}$ can interact with parties in $\mathcal{P}$ and the environment $\mathcal{Z}$. Namely, $\mathcal{Z}$ sends L_{corr} to $\mathcal{H}$ at the beginning or the execution. In this work, we allow $\mathcal{H}$ to run in super-polynomial time w.r.t. the security parameter λ. At a high level, $\mathcal{F}^{f,R}_{\mathsf{vmpc}}(\mathcal{P})$ interacts with the ideal adversary Sim as follows:

– At the **Initialize** phase, it waits for the servers and clients to be ready for the VMPC execution.

– At the **Input** phase, it receives the user's inputs. It leaks the input of U_ℓ to the adversary only if (i) all servers are corrupted or (ii) the client C_ℓ of U_ℓ is corrupted. If neither (i) nor (ii) holds, then $\mathcal{F}^{f,R}_{\mathsf{vmpc}}(\mathcal{P})$ only reveals whether U_ℓ abstained from the execution.

– At the **Compute** phase, upon receiving all user's inputs denoted as vector $\mathbf{x} \in X^n$, it computes the output value $y = f(\mathbf{x})$.

– At the **Verify** phase, upon receiving a verification request from V (which is a dummy party here), the functionality is responsible for playing the role of an "ideal verifier" for every user U_ℓ. On the other hand, Sim sends to $\mathcal{F}^{f,R}_{\mathsf{vmpc}}(\mathcal{P})$ an adversarial (hence, not necessarily meaningful) output value $\tilde{y}$ for the VMPC execution for U_ℓ. Then, $\mathcal{F}^{f,R}_{\mathsf{vmpc}}(\mathcal{P})$'s verification verdict w.r.t. U_ℓ will depend on the interaction with Sim and potentially the relation of $y, \tilde{y}$ w.r.t. R. We stress that $\mathcal{F}^{f,R}_{\mathsf{vmpc}}(\mathcal{P})$ will consider $\tilde{y}$ only if (a) all servers are corrupted, or (b) an honest user's client is corrupted[6]. If this is not the case, then it will always send the actual computed value y to U_ℓ and its verification verdict will not depend on R, which is in line with the standard notion of MPC correctness. The functionality $\mathcal{F}^{f,R}_{\mathsf{vmpc}}(\mathcal{P})$ is presented in Figure 3.

Real world setting. In the real world setting, all the entities specified in the set $\mathcal{P}$ are involved in an execution of a VMPC scheme $\Pi = (\textbf{Initialize}, \textbf{Input}, \textbf{Compute}, \textbf{Verify})$ in the presence of functionalities $\mathcal{G}_{\mathsf{BB}}$ and $\mathcal{F}_{\mathsf{sc}}$. As in the ideal world, the environment $\mathcal{Z}$ provides the inputs and determines the corruption subset $L_{\mathsf{corr}} \subset \mathcal{P}$. $\mathcal{Z}$ will also send L_{corr} to $\mathcal{H}$ at the beginning of the execution. During **Initialize**, the servers interact with the users' clients. During the **Input** protocol, every honest user U_ℓ engages by providing her private input x_ℓ via C_ℓ and obtaining her individual audit data α_ℓ. The execution is run in the presence of a PPT adversary $\mathcal{A}$ that observes the network traffic and corrupts the parties specified in L_{corr}.

VMPC definition. As in the $\mathcal{H}$-EUC framework [18], we consider an environment $\mathcal{Z}$ that provides inputs to all parties, interacts with helper $\mathcal{H}$ and schedules the execution. In the ideal world setting, $\mathcal{Z}$ outputs the bit $\mathsf{EXEC}^{\mathcal{F}^{f,R}_{\mathsf{vmpc}}(\mathcal{P})}_{\mathsf{Sim},\mathcal{Z},\mathcal{H}}(\lambda)$, and in the real world the bit $\mathsf{EXEC}^{\mathcal{P},\Pi^{\mathcal{G}_{\mathsf{BB}},\mathcal{F}_{\mathsf{sc}}}}_{\mathcal{A},\mathcal{Z},\mathcal{H}}(\lambda)$. Security is defined as follows:

Definition 6. *Let $f : X^n \longrightarrow Y$ be an evaluation function and $R \subseteq Y \times Y$ be a binary relation. Let $\mathcal{H}$ be a helper functionality. We say that a VMPC scheme $\Pi^{\mathcal{G}_{\mathsf{BB}},\mathcal{F}_{\mathsf{sc}}}$ operating with the parties in $\mathcal{P}$, $\mathcal{H}$-*EUC *realizes* $\mathcal{F}^{f,R}_{\mathsf{vmpc}}(\mathcal{P})$ *with error ϵ, if for every PPT adversary $\mathcal{A}$ there is an ideal PPT simulator* Sim *such that for every PPT environment $\mathcal{Z}$, it holds that*

$$\left| \Pr\left[\mathsf{EXEC}^{\mathcal{F}^{f,R}_{\mathsf{vmpc}}(\mathcal{P})}_{\mathsf{Sim},\mathcal{Z},\mathcal{H}}(\lambda) = 1\right] - \Pr\left[\mathsf{EXEC}^{\mathcal{P},\Pi^{\mathcal{G}_{\mathsf{BB}},\mathcal{F}_{\mathsf{sc}}}}_{\mathcal{A},\mathcal{Z},\mathcal{H}}(\lambda) = 1\right] \right| < \epsilon\ .$$

Strength of our VMPC security model. Based on the description of $\mathcal{F}^{f,R}_{\mathsf{vmpc}}$, the private input x_ℓ of an honest user U_ℓ is leaked if her client C_ℓ is corrupted, or if all servers are malicious, so in our VMPC model, the honest users' clients and at least one server must be non-corrupted for privacy. For integrity, we require that the verifier remains honest, while $\mathcal{G}_{\mathsf{BB}}$ captures the notion of a consistent and public bulletin board. We informally argue that these requirements are essential for VMPC feasibility, at least for meaningful cases of functions and relations.

[6] In case an honest user's client is corrupted, an "input replacement" attack can take place which makes it impossible to deliver (the true output) y to the user.

The functionality operates with the following parties $\mathcal{P} = \mathcal{U} \cup \mathcal{C} \cup \mathcal{S} \cup \{V\}$. The set $L_{\mathsf{corr}} \subset \mathcal{P}$ contains all corrupted parties.

Initialize.

– It sets its status to 'init' and initializes four lists $L_{\mathsf{start}}, L_{\mathsf{comp}}, L_{\mathsf{cast}}$ and L_{ready} as empty and a list L_{in} as $\langle (U_\ell, \cdot) \rangle_{U_\ell \in \mathcal{U}}$.

– Upon receiving $(\textsc{Start}, \mathsf{sid})$ from $S_i \in \mathcal{S}$, if its status is 'init', then it updates $L_{\mathsf{start}} \leftarrow L_{\mathsf{start}} \cup \{\, S_i \,\}$. If $|L_{\mathsf{start}}| = k$, it sets the status to 'input'.

– Upon receiving $(\textsc{Ready}, \mathsf{sid})$ from C_ℓ, if its status is 'init', then it sends public delayed output $(\textsc{Ready}, \mathsf{sid})$ to Sim and adds C_ℓ to L_{ready}.

Input.

– Upon receiving $(\textsc{Cast}, \mathsf{sid}, x_\ell)$ from U_ℓ, if (i) the status is 'input', and (ii) $C_\ell \in L_{\mathsf{corr}}$ or $\{S_1, \ldots, S_k\} \subseteq L_{\mathsf{corr}}$, then it sends $(\mathsf{sid}, \mathsf{cast}, U_\ell, x_\ell)$ to Sim. Otherwise, it sends $(\mathsf{sid}, \mathsf{cast}, U_\ell, (x_\ell \overset{?}{=} \text{'}\mathsf{abstain}\text{'}))$ to Sim. If the status is 'input' and the entry in L_{in} indexed by U_ℓ is $(U_\ell, \cdot)$, then it updates the entry as (U_ℓ, x_ℓ).

– Upon receiving $(\textsc{Record}, \mathsf{sid}, U_\ell, \tilde{x}_\ell)$ from Sim, if (i) the status is 'input', and (ii) $(U_\ell, \cdot) \notin L_{\mathsf{cast}}$, then

∘ If $C_\ell \in L_{\mathsf{corr}}$, then it adds $(U_\ell, \tilde{x}_\ell)$ to L_{cast}.

∘ If (a) $C_\ell \notin L_{\mathsf{corr}}$, (b) there is a record $(U_\ell, x_\ell) \in L_{\mathsf{in}}$ and (c) $C_\ell \in L_{\mathsf{ready}}$ or $x_\ell = \mathsf{abstain}$, then it adds (U_ℓ, x_ℓ) to L_{cast}.

– Upon receiving $(\textsc{Compute}, \mathsf{sid})$ from $S_i \in \mathcal{S}$, if its status is 'input', then it updates $L_{\mathsf{comp}} \leftarrow L_{\mathsf{comp}} \cup \{\, S_i \,\}$. If $|L_{\mathsf{comp}}| = k$, it sets the status to '$\mathsf{compute}$'. For every U_ℓ s.t. there is no record in L_{cast}, it adds $(U_\ell, \mathsf{abstain})$ to L_{cast}.

Compute.

– If status is '$\mathsf{compute}$' and L_{cast} contains records for all users $U_1, \ldots, U_n$, it computes $y \leftarrow f\big(\langle x_\ell \rangle_{(U_\ell, x_\ell) \in L_{\mathsf{cast}}}\big)$ and sends $(\textsc{Output}, \mathsf{sid}, y)$ to Sim.

– Upon receiving $(\textsc{Audit}, \mathsf{sid})$ from Sim, it sets the status to 'audit'.

Verify.

– Upon receiving $(\textsc{Verify}, \mathsf{sid})$ from V, if the status is 'audit', then it sends $(\textsc{Verify}, \mathsf{sid})$ to Sim.

– Upon receiving $(\textsc{Verify\_response}, \mathsf{sid}, U_\ell, \tilde{y}, \tilde{v})$ from Sim, if the status is 'audit':

(1) If (i) $\tilde{v} = 1$, and (ii) $V \notin L_{\mathsf{corr}}$ then

• If there exists an $S_i \notin L_{\mathsf{corr}}$ and for all ℓ' such that $U_{\ell'} \notin L_{\mathsf{corr}}$ it holds that $C_{\ell'} \notin L_{\mathsf{corr}}$, then it sends $(\textsc{Result}, \mathsf{sid}, y, 1)$ to U_ℓ.

• Else, (all servers are corrupted, or there is an honest $U_{\ell'}$ with a corrupted $C_{\ell'}$)

∘ If $(y, \tilde{y}) \in R$, then it sends $(\textsc{Result}, \mathsf{sid}, \tilde{y}, 1)$ to U_ℓ.

∘ If $(y, \tilde{y}) \notin R$, then it sends $(\textsc{Result}, \mathsf{sid}, \tilde{y}, 0)$ to U_ℓ.

(2) Else if (i) $\tilde{v} = 0$ or (ii) $V \in L_{\mathsf{corr}}$, then it sends $(\textsc{Result}, \mathsf{sid}, \tilde{y}, \tilde{v})$.

Fig. 3: The ideal VMPC functionality $\mathcal{F}^{f,R}_{\mathsf{vmpc}}(\mathcal{P})$.

Clearly, since the users communicate with the servers only via her client, the user has to provide her input to the client which has to be trusted for privacy. Besides, if the adversary can corrupt all the servers, then it can completely run the **Compute** protocol and along with the environment, schedule the evaluation of f that, in general, may leak information on individual inputs that Sim cannot infer just by receiving the evaluation of f on the entire input vector.

Furthermore, if the real world verifier is malicious, then it can provide arbitrary verdicts regardless of the "verification rules" imposed by R, which rules are respected by $\mathcal{F}^{f,R}_{\mathsf{vmpc}}(\mathcal{P})$ in the ideal world (the same would hold even we considered multiple verifiers per user). Finally, in case of no consistent BB, since the communication between parties is not assumed authenticated, an adversary can disconnect the parties separating them into disjoint groups, and provide partial and mutually inconsistent views of the VMPC execution per group. For more details, we refer to Barak *et al.* [3] and Supplementary Material D, where we discuss the strength of our model w.r.t. the server, client, and verifier corruption.

6 Spreading Relations

In this section, we study the characteristics that a function $f : X^n \longrightarrow Y$ must have w.r.t. some relation $R \subseteq \mathsf{Img}[f] \times \mathsf{Img}[f]$ to be realized by a VMPC scheme. Recall that in our setting, all entities capable of performing cryptographic operations might be corrupted and only a subset of users is honest. This requirement poses limitations not present in other security models (e.g. [4]), where auditable/verifiable MPC is feasible for a large class of functions (arithmetic circuits) given (i) the existence of a trusted randomness source or a random oracle or (ii) the fact that both the honest user and her client are considered as one non-corrupted entity. As a consequence, for some evaluation function f and binary relation R, if VMPC realization is feasible, then this is due to the nature of the users' engagement in the VMPC execution. Namely, we consider that the users interact using some randomness that implies a level of *unpredictability* in the eyes of the attacker that prevents end-to-end verifiability (as determined by relation R) or secrecy from being breached. Naturally, this engagement results in a security error that strongly depends on (i) *the number of honest users* whose inputs are attacked by the adversary and (ii) *the user min entropy* κ. On the contrary, it is plausible that if an adversary controlling the entire execution can guess all the users' coins, then this execution is left defenseless against the adversary's attacks. As mentioned in Section 5, the possible values for κ remain at a "human level", in the sense that the randomness r_ℓ of U_ℓ can be guessed with good probability. Typically, we assume that $2^{-\kappa}$ is non-negligible in the security parameter λ by setting $\kappa = O(\mathsf{log}\lambda)$.

We view the sets X^n and Y as metric spaces equipped with metrics d_{X^n} and d_Y respectively. For the domain X^n, we select the metric that provides an estimation of the number of honest users that have been attacked, i.e. their inputs are modified by the real world adversary. So, we fix d_{X^n} as the metric Dcr_n that counts the number of vector elements that two inputs $\mathbf{x} = (x_1, \ldots, x_n), \mathbf{x}' = (x'_1, \ldots, x'_n)$ differ. Formally, $\mathsf{Dcr}_n(\mathbf{x}, \mathbf{x}') = \big|\{\, \ell \in [n] \mid x_\ell \neq x'_\ell \,\}\big|$.

We examine feasibility of realizing $\mathcal{F}^{f,R}_{\mathsf{vmpc}}$ w.r.t. f, R according to the following reasoning: assuming that cryptographic security holds, then an adversarial input that has some distance δ w.r.t. Dcr_n from the honest inputs cannot cause a significant divergence y' from the actual evaluation $y = f(\mathbf{x})$. Here, divergence is interpreted as the case where y, y' are not in some fixed relation R. For instance, if divergence means that the deviation from the actual evaluation is no more than δ, this can be expressed as y, y' not being in the *bounded distance relation* R_δ defined as follows:

$$R_\delta := \{(z, z') \in Y \times Y \mid \mathrm{d}_Y(z, z') \leq \delta\} \ . \tag{2}$$

An interesting class of evaluation functions that can be realized in an VMPC manner w.r.t. R_δ are the ones that satisfy some *relaxed isometric property*, thus inherently preventing evaluation from "large" deviation blow ups when the distance between honest and adversarial inputs is bounded, as specified by Eq. (2) for some positive value δ. One noticeable example are the *Lipschitz functions*; namely, for some $L > 0$, if the evaluation function $f : X^n \longrightarrow Y$ is L*-Lipschitz*, then for every $\mathbf{x}, \mathbf{x}' \in X^n$ it holds that $\mathrm{d}_Y\big(f(\mathbf{x}), f(\mathbf{x}')\big) \leq L \cdot \mathsf{Dcr}_n\big(\mathbf{x}, \mathbf{x}'\big)$.

Thus, in the case of an L-Liptshitz function f and bounded distance relation R_δ, the following condition holds:

$$\forall \mathbf{x}, \mathbf{x}' \in X^n : \ \mathsf{Dcr}_n(\mathbf{x}, \mathbf{x}') \leq \delta/L \Rightarrow R_\delta\big(f(\mathbf{x}), f(\mathbf{x}')\big) \ .$$

In general, the above condition implies that the ideal functionality $\mathcal{F}^{f,R}_{\mathsf{vmpc}}(\mathcal{P})$ will accept a simulation when the adversarial value y' *can be derived by an input vector* that is no more than δ-far from the actual users' inputs. This interesting property fits perfectly with our intuition of VMPC realization and captures Lipschitz functions and bounded distance relations as special case. Based on the above, we introduce the notion of *spreading relations* as follows.

Definition 7 (Spreading relation). *Let* (X^n, Dcr_n) *and* (Y, d_Y) *be metric spaces,* $f : X^n \longrightarrow Y$ *be a function and* δ *be a non-negative real value. We say that* $R \subseteq \mathsf{Img}[f] \times \mathsf{Img}[f]$ *is a* δ-spreading relation over $\mathsf{Img}[f]$*, if for every* $\mathbf{x}, \mathbf{x}' \in X^n$ *it holds that*

$$\mathsf{Dcr}_n(\mathbf{x}, \mathbf{x}') \leq \delta \Rightarrow R\big(f(\mathbf{x}), f(\mathbf{x}')\big) \ .$$

The breadth of VMPC feasibility. Given Definition 7, we formally explore the boundaries of VMPC feasibility given some fixed values κ, δ. Intuitively, we show that if f is symmetric[7], then VMPC realization with a small (typically $\mathsf{negl}(\delta)$) error is infeasible when R is not a δ-spreading relation over $\mathsf{Img}[f]$, or if the users engage in the VMPC execution in a "deterministic way" (i.e., $\kappa = 0$). A detailed discussion and a proof sketch can be found in Supplementary Material E.

Theorem 3. *Let* $f : X^n \longrightarrow Y$ *be a symmetric function,* $R \subseteq \mathsf{Img}[f] \times \mathsf{Img}[f]$ *be a binary relation and* κ, δ *be non-negative values, where* $\delta \leq \frac{n}{2}$*. Then, one of the following two conditions holds:*

(1) *R is a δ-spreading relation over* $\mathsf{Img}[f]$.

[7] $f(x_1, \ldots, x_n)$ is symmetric iff it is unchanged by any permutation of its variables.

(2) For every VMPC scheme $\Pi^{\mathcal{G}_{\mathsf{BB}},\mathcal{F}_{\mathsf{sc}}}$ with parties in $\mathcal{P} = \{U_1, \ldots, U_n\} \cup \{C_1, \ldots, C_n\} \cup \{S_1, \ldots, S_k\} \cup \{V\}$ and user min entropy κ, and every helper $\mathcal{H}$, there is a negligible function ϵ and a non-negligible function γ such that $\Pi^{\mathcal{G}_{\mathsf{BB}},\mathcal{F}_{\mathsf{sc}}}$ does not $\mathcal{H}$-EUC realize $\mathcal{F}_{\mathsf{vmpc}}^{f,R}(\mathcal{P})$ with error less than $\min\{2^{-\kappa\delta} - \epsilon(\lambda), \gamma(\lambda)\}$.

7 Constructing VMPC from CVZK

A number of efficient practical MPC protocols [12, 57, 29, 28] have been proposed in the pre-processing model. Such protocols consist of two phases: *offline* and *online*. During the *offline* phase, the MPC parties jointly compute authenticated correlated randomness, which typically is independent of the parties' inputs. During the *online* phase, the correlated randomness is consumed to securely evaluate the MPC function over the parties' inputs. Our VMPC construction follows the same paradigm as [4]. Our main challenge is to transform a publicly audible MPC to a VMPC *without* a trusted setup.

Our construction utilizes (i) a perfectly binding homomorphic commitment that is secure against helper-aided PPT adversaries (cf. C.2), (ii) a *dual-mode homomorphic commitment* DC, which allows for two ways to choose the commitment key s.t. the commitment is either perfectly binding or equivocal (constructed in F.1), (iii) a *Σ-protocol for Beaver triples*, (cf. F.2) and (iv) CVZK proofs that derive from compiling straight-line simulatable ZK proofs for **NP** languages (cf. F.4) via our CVZK construction from Section 4. Note that plain ZK does not comply with the VMPC corruption model, as all servers and clients can be corrupted and each user has limited entropy. Additionally, our protocol utilizes a secure channel functionality $\mathcal{F}_{\mathsf{sc}}$ between human users U_ℓ and their local clients C_ℓ; and an authenticated channel functionality $\mathcal{F}_{\mathsf{auth}}$ between human users U_ℓ and verifier V. Both channels can be instantiated from physical world, such as isolated voting rooms and trusted mailing service. To provide intuition, we first provide a construction for the single-server setting.

7.1 Single-server VMPC

As a warm-up , we present the simpler case of a single MPC server S. In this setting, no privacy can be guaranteed when S is corrupted, yet end-to-end verifiability should remain, since the property should hold even if *all servers* are corrupted. For simplicity, by using CVZK to prove a statement, we mean that the prover (server) runs $\mathsf{CVZK.Prv}_1$ to generate the first move of the CVZK proof and posts it on BB (formalized as $\mathcal{G}_{\mathsf{BB}}$ in Fig. 6) during the **Initialize** phase. Each user then acts as a CVZK verifier to generate and post a coin on the BB at **Input** phase. The prover uses $\mathsf{CVZK.Prv}_2$ to complete the proof by posting the third move of the CVZK proof to the BB at the **Compute** phase. At **Verify**, anyone can check the CVZK transcripts posted on the BB.

– At the **Initialize** phase, S first generates a perfectly binding commitment key of the dual-mode homomorphic commitment as $\mathsf{ck} \leftarrow \mathsf{DC.Gen}(1^\lambda)$ which posts on the BB and shows that ck is a binding key using CVZK. Then, S generates and commits to two random numbers $r_\ell^{(0)}, r_\ell^{(1)} \in \mathbb{Z}_p$ to the BB for each user U_ℓ, $\ell \in [n]$. Denote the corresponding commitments as $c_\ell^{(0)}$ and $c_\ell^{(1)}$. Furthermore, S generates sufficiently many random Beaver triples (depending on the multiplication gates of the circuit to be evaluated), i.e., triples $(a, b, c) \in (\mathbb{Z}_p)^3$ such that $c = a \cdot b$, and then commits the triples to the BB by showing their correctness using the CVZK compiled from the Σ-protocol for Beaver triples described in F.2. For each user U_ℓ, $\ell \in [n]$, S sends $r_\ell^{(0)}$ and $r_\ell^{(1)}$ to her client C_ℓ.

– At the **Input** phase, C_ℓ sends (displays) $r_\ell^{(0)}$ and $r_\ell^{(1)}$ to U_ℓ. Assume U_ℓ's input is x_ℓ. U_ℓ randomly picks $b_\ell \leftarrow \{0, 1\}$ and computes $\delta_\ell = x_\ell - r_\ell^{(b_\ell)}$[8]. Then, U_ℓ sends (b_ℓ, δ_ℓ) to C_ℓ, which in turn posts $(U_\ell, \delta_\ell, b_\ell)$ to the BB, where U_ℓ is the user ID. Finally, U_ℓ obtains $(b_\ell, \delta_\ell, r_\ell^{(1-b_\ell)})$ as her individual audit data α_ℓ.

– At the **Compute** phase, S fetches posted messages from the BB. For $\ell \in [n]$, S sets $c_\ell \leftarrow c_\ell^{(b_\ell)} \cdot \mathsf{DC.Com}_{\mathsf{ck}}(\delta_\ell; \mathbf{0})$ and opens $c_\ell^{(1-b_\ell)}$ to the BB (note that c_ℓ commits to x_ℓ). S follows the arithmetic circuit to evaluate $f(x_1, \ldots, x_n)$ using $(c_1, \ldots, c_n)$ as the input commitments. Specifically, (i) for addition gate $z = x + y$, S uses homomorphic property to set the commitment of z as $\mathsf{DC.Com}_{\mathsf{ck}}(x) \cdot \mathsf{DC.Com}_{\mathsf{ck}}(y)$; (ii) for multiplication gate $z = x \cdot y$, S needs to consume a pre-committed random Beaver triple. Denote the commitments of x and y as X and Y, respectively and the triple commitments as (A, B, C) which commit to a, b, c s.t. $a \cdot b = c$. Then, S opens the commitment X/A as α and Y/B as β to the BB. It then sets the commitment of z as $C \cdot B^\alpha \cdot A^\beta \cdot \mathsf{DC.Com}_{\mathsf{ck}}(\alpha \cdot \beta)$. By homomorphic property, it is easy to see that $z = x \cdot y$. Finally, S opens the commitments corresponding to the output gate(s) of the arithmetic circuit as the final result.

– At the **Verify** phase, V requests and receives the individual audit data $\{\alpha_\ell\}_{\ell \in [n]}$ from each user U_ℓ, $\ell \in [n]$, via $\mathcal{F}_{\mathsf{auth}}$. First, V parses $\alpha_\ell = (b_\ell, \delta_\ell, r_\ell^{(1-b_\ell)})$, for $\ell \in [n]$. Next, V fetches all the transcript from the BB, and it executes the following steps: (1) it checks that the posted b_ℓ on the BB match the ones in α_ℓ; (2) it verifies that the openings of all the commitments are valid; (3) it verifies that all the CVZK proofs are valid; (4) it re-computes the arithmetic circuit using the commitments and openings posted on the BB to verify the computation correctness. If all checks are successful, V sets the verification bit $v := 1$, else it sets $v := 0$. Finally, it sends the opening of the result commitment (i.e., $f(x_1, \ldots, x_n)$) along with v to every user U_ℓ, $\ell \in [n]$.

7.2 Security of the single-server construction

We provide an informal discussion on the security of the single-server construction in terms of privacy and end-to-end verifiability.

Privacy. The single-server VMPC construction preserves user U_ℓ's privacy when the server S and C_ℓ are honest. In particular, since the underlying com-

[8] Note that this step requires the "human" user to perform some linear operation in $\mathbb{Z}_p$. If we want to avoid *any* type of computation in the user side (apart from coin-flipping), then the client can also send a pre-computed lookup table for all δ_ℓ (assuming that the user input space is polynomial).

mitment scheme is computationally hiding under the adaptively secure DDH assumption, all the posted commitments to values X/A and Y/B leak no information (up to a $\mathsf{negl}(\lambda)$ error) about the users' inputs to a PPT adversary with access to the helper. Furthermore, while computing the multiplication gates, the openings have uniform distribution, as the plaintext is masked by a random group element.

End-to-end verifiability. Let f be an evaluation function and R be a δ-spreading relation over $\mathsf{Img}[f]$ (cf. Definition 7), where $\delta \geq 0$ is an integer. We informally discuss how the single-server VMPC protocol achieves end-to-end verifiability w.r.t. R, with error that is negligible in λ and δ. Recall that the soundness of the underlying CVZK proofs and the binding property of the (dual-mode) commitment scheme holds even against unbounded adversaries, therefore argumentation for end-to-end verifiability captures the case of helper-aided adversaries. We stress that the argumentation can extend to the multiple server case, since the adversary is allowed to corrupt all servers as a single malicious authority. Assume that the adversary $\mathcal{A}$ corrupts the MPC server, all users' clients and no more than $n^{1-\frac{1}{\gamma}}/\log^3 n$ users. First, we note that if $\mathcal{A}$ additionally corrupts the verifier V, we can construct a simple simulator that engages with $\mathcal{A}$ by playing the role of honest users and simply forwards the malicious response of V to $\mathcal{F}^{f,R}_{\mathsf{vmpc}}(\mathcal{P})$ along with the adversarial tally y'. For the more interesting case where V is honest, we list the types of attacks that $\mathcal{A}$ may launch below:

– *Commitment attack:* $\mathcal{A}$ attempts to open some commitment c of a message m, to a value $m' \neq m$. By the perfect binding property of ElGamal commitment, this attack has zero success probability.

– *Soundness attack:* $\mathcal{A}$ attempts to convince the verifier of an invalid CVZK proof. By the $\left(n^{1-\frac{1}{\gamma}}/\log^3 n, \mathsf{negl}(\lambda)\right)$-crowd-verifiable soundness of our CVZK compiler (cf. Theorem 2), $\mathcal{A}$ has $\mathsf{negl}(\lambda)$ probability of success in such an attack.

– *Client attack:* by corrupting the client C_ℓ of U_ℓ, $\mathcal{A}$ provides U_ℓ with a pair of random values $(\hat{r}^{(0)}_\ell, \hat{r}^{(1)}_\ell)$, where one component $\hat{r}^{(b^*)}_\ell$ is different than $r^{(b^*)}_\ell$ in the pair $(r^{(0)}_\ell, r^{(1)}_\ell)$ committed to BB (formalized as $\mathcal{G}_{\mathsf{BB}}$ in Fig. 6). Hence, if $\mathcal{A}^*$ guesses the coin of U_ℓ correctly (i.e. $b^* = b_\ell$), then it can perform the VMPC execution by replacing U_ℓ's input x_ℓ with input $x^*_\ell = x_\ell + \left(\hat{r}^{(b^*)}_\ell - r^{(b^*)}_\ell\right)$ without being detected. Given that U_ℓ flips a fair coin, this attack has 1/2 success probability.

Observe that the above list is complete; if none of the above attacks happen, then by the properties of the secret sharing scheme, $\mathcal{A}$ can not tamper the VMPC computation on the consistent BB without being detected.

Leaving aside the $\mathsf{negl}(\lambda)$ cryptographic error inserted by combinations of commitment and soundness attacks, the adversary's effectiveness relies on the scale of client attacks that it can execute. If it performs more than δ client attacks, then by the description of client attacks, V will detect and reject with at least $1 - 2^{-\delta}$ probability. So, with at least $1 - 2^{-\delta}$ probability, a simulator playing the role of the (honest) verifier will also send a reject message ($\tilde{v} = 0$) for every honest user to $\mathcal{F}^{f,R}_{\mathsf{vmpc}}(\mathcal{P})$ and indistinguishability is preserved.

On the other hand, if $\mathcal{A}$ performs less than δ client attacks, then the actual input $\mathbf{x}$ and the adversarial one $\mathbf{x}'$ are δ-close w.r.t. $\mathsf{Dcr}_n(\cdot,\cdot)$. Since the relation R is δ-spreading, we have that $\big(f(\mathbf{x}), f(\mathbf{x}')\big) \in R$ holds. So, when the simulator plays the role of the (honest) verifier that accepts, it sends an accept message ($\tilde{v} = 1$) for every honest user to $\mathcal{F}_{\mathsf{vmpc}}^{f,R}(\mathcal{P})$ which in turn will also accept (since $\big(f(\mathbf{x}), f(\mathbf{x}')\big) \in R$ holds). Besides, $\mathcal{F}_{\mathsf{vmpc}}^{f,R}(\mathcal{P})$ will reject whenever the simulator sends a reject message, hence, indistinguishability is again preserved.

We conclude that the single-server VMPC scheme achieves end-to-end verifiability with overall error $2^{-\delta} + \mathsf{negl}(\lambda)$.

Observe that the above list is complete; if none of the above attacks happen, then by the properties of the secret sharing scheme, $\mathcal{A}$ can not tamper the VMPC computation on the consistent BB without being detected.

Leaving aside the $\mathsf{negl}(\lambda)$ cryptographic error inserted by combinations of commitment and soundness attacks, the adversary's effectiveness relies on the scale of client attacks that it can execute. If it performs more than δ client attacks, then by the description of client attacks, V will detect and reject with at least $1 - 2^{-\delta}$ probability. So, with at least $1 - 2^{-\delta}$ probability, a simulator playing the role of the (honest) verifier will also send a reject message ($\tilde{v} = 0$) for every honest user to $\mathcal{F}_{\mathsf{vmpc}}^{f,R}(\mathcal{P})$ and indistinguishability is preserved.

On the other hand, if $\mathcal{A}$ performs less than δ client attacks, then the actual input $\mathbf{x}$ and the adversarial one $\mathbf{x}'$ are δ-close w.r.t. $\mathsf{Dcr}_n(\cdot,\cdot)$. Since the relation R is δ-spreading, we have that $\big(f(\mathbf{x}), f(\mathbf{x}')\big) \in R$ holds. So, when the simulator plays the role of the (honest) verifier that accepts, it sends an accept message ($\tilde{v} = 1$) for every honest user to $\mathcal{F}_{\mathsf{vmpc}}^{f,R}(\mathcal{P})$ which in turn will also accept (since $\big(f(\mathbf{x}), f(\mathbf{x}')\big) \in R$ holds). Besides, $\mathcal{F}_{\mathsf{vmpc}}^{f,R}(\mathcal{P})$ will reject whenever the simulator sends a reject message, hence, indistinguishability is again preserved.

We conclude that the single-server VMPC scheme achieves end-to-end verifiability with overall error $2^{-\delta} + \mathsf{negl}(\lambda)$.

7.3 Extension to multi-server VMPC

The single-server VMPC can be naturally extended to a multi-server version by secret-sharing the server's state. The protocol is similar to BDO [4] and SPDZ [29, 28]. However, all the underlying ZK proofs need to be compiled in CVZK. Let $\mathbb{Z}_p$ be the message space of the secret shared values. A secret shared value $x \in \mathbb{Z}_p$ is represented by $\langle x \rangle := (x^{(1)}, \ldots, x^{(k)})$, where each MPC server S_j, $j \in [k]$, holds the random share $x^{(j)}$ such that $x = \sum_{j=1}^{k} x^{(j)}$. Each shared value $\langle x \rangle$ is associated with a (secret shared) fresh randomness $\rho \in (\mathbb{Z}_p)^k$ and a commitment $\mathsf{DC.Com}_{\mathsf{ck}}(x; \rho)$. More specifically, a *publicly traceable* secret shared value x is $[\![\cdot]\!]$-represented as $[\![x]\!] := (\langle x \rangle, \langle \rho \rangle, \mathsf{DC.Com}_{\mathsf{ck}}(x; \rho))$, where shares of x and ρ are held by the MPC servers. $\mathsf{DC.Com}_{\mathsf{ck}}(x; \rho)$ is posted to $\mathcal{G}_{\mathrm{BB}}$. In the following, we provide a detailed description of the VMPC data representation.

Data representation. The main difference between our data representation and the SPDZ (and its variants) is that we do not attach linear MAC to the shared data. This is mainly because we aim to achieve verifiability when all the MPC servers are corrupted, and all the shared values including the MAC key are known to the adversary in that extreme case. Hence, the correctness of the online phase execution cannot be ensured by checking MACs. Instead, similar as in [5], all the shared values have also a corresponding commitment posted on $\mathcal{G}_{\text{BB}}$; therefore, the auditor can run the exact MPC online circuit over the commitments and check if the opening of the final commitment matches the MPC output. The difference between [5] and ours is that we also commit individual shares. More specifically, let the message space of the shared value be $\mathbb{Z}_p$. A secret shared value $x \in \mathbb{Z}_p$ is represented by

$$\langle x \rangle := (x^{(1)}, \ldots, x^{(k)}) \enspace ,$$

where each MPC server S_j, $j \in [k]$, holds the random share $x^{(j)}$ such that $x = \sum_{j=1}^{k} x^{(j)}$. Each shared value $\langle x \rangle$ associated with a (secret shared) fresh randomness $\rho(x) \in \mathbb{Z}_p$ and a commitment $\mathsf{Com}_{\mathsf{ck}}(x; \rho(x))$. More specifically, a publicly traceable secret shared value x is $[\![\cdot]\!]$-represented as below:

$$[\![x]\!] := (\langle x \rangle, \langle \rho(x) \rangle, \mathsf{Com}_{\mathsf{ck}}(\langle x \rangle; \langle \rho(x) \rangle), \mathsf{Com}_{\mathsf{ck}}(x; \rho(x)) \enspace ,$$

where shares of x and $\rho(x)$ are held by the MPC servers, and for notation simplicity, we use the notation $\mathsf{Com}_{\mathsf{ck}}(\langle x \rangle; \langle \rho(x) \rangle)$ to denote a vector of commitments of the individual shares, i.e.,

$$\mathsf{Com}_{\mathsf{ck}}(x^{(1)}; \rho(x^{(1)})), \ldots, \mathsf{Com}_{\mathsf{ck}}(x^{(k)}; \rho(x^{(k)}))$$

$\mathsf{Com}_{\mathsf{ck}}(\langle x \rangle; \langle \rho(x) \rangle)$ and $\mathsf{Com}_{\mathsf{ck}}(x; \rho(x))$ are posted to $\mathcal{G}_{\text{BB}}$. Due to homomorphic property, we have

$$\mathsf{Com}_{\mathsf{ck}}(x; \rho(x)) = \prod_{i=1}^{k} \mathsf{Com}_{\mathsf{ck}}(x^{(i)}; \rho(x^{(i)})) \enspace .$$

Information wise, $\mathsf{Com}_{\mathsf{ck}}(x; \rho(x))$ is redundant, but we keep it for conceptual clarity. It is easy to see that $[\![\cdot]\!]$ shares are linearly homomorphic:

$$\begin{aligned}
[\![x]\!] + [\![y]\!] :&= \big(\langle x \rangle + \langle y \rangle, \langle \rho(x) \rangle + \langle \rho(y) \rangle, \\
&\quad \mathsf{Com}_{\mathsf{ck}}(\langle x \rangle; \langle \rho(x) \rangle) \cdot \mathsf{Com}_{\mathsf{ck}}(\langle y \rangle; \langle \rho(y) \rangle) \\
&\quad \mathsf{Com}_{\mathsf{ck}}(x; \rho(x)) \cdot \mathsf{Com}_{\mathsf{ck}}(y; \rho(y))\big) \\
[\![x]\!] + c :&= \big(\langle x \rangle + c, \langle \rho(x) \rangle, \big(\mathsf{Com}_{\mathsf{ck}}(x^{(1)}; \rho(x^{(1)}) \cdot \mathsf{Com}_{\mathsf{ck}}(c; 0), \\
&\quad \mathsf{Com}_{\mathsf{ck}}(x^{(2)}; \rho(x^{(2)}), \ldots, \mathsf{Com}_{\mathsf{ck}}(x^{(k)}; \rho(x^{(k)}))\big), \\
&\quad \mathsf{Com}_{\mathsf{ck}}(x; \rho(x)) \cdot \mathsf{Com}_{\mathsf{ck}}(c; 0)\big) \\
[\![x]\!] \cdot c :&= \big(\langle x \rangle \cdot c, \langle \rho(x) \rangle \cdot c, \\
&\quad (\mathsf{Com}_{\mathsf{ck}}(\langle x \rangle; \langle \rho(x) \rangle))^c, (\mathsf{Com}_{\mathsf{ck}}(x; \rho(x)))^c\big)
\end{aligned}$$

The offline phase. Similar to all the MPCs with preprocessing, our VMPC also uses offline phase to generate sufficiently many correlated randomness. More precisely, we need to generate shared random Beaver triples and shared random values. The main differences between $\mathcal{F}_{\mathsf{V.Offline}}$ and the ones used in SPDZ and its variants are (i) The MAC is removed from all the shares, and (ii) $\mathcal{F}_{\mathsf{V.Offline}}$ has to be crowd verifiable. The functionality is depicted in Fig. 4. The MPC servers initialize the functionality by sending $(\mathsf{Init}, sid, n_1, n_2)$ to $\mathcal{F}_{\mathsf{V.Offline}}$. If all the servers are corrupted, the functionality let the adversary Sim choose the random coin Ω and use it to generate the commitment key as $\mathsf{ck} \leftarrow \mathsf{Gen}_{\mathsf{Com}}(\mathsf{Param}_{\mathsf{gp}}; \Omega)$. Otherwise, $\mathcal{F}_{\mathsf{V.Offline}}$ generate the commitment key with a freshly sampled random coin. After that, it sends the commitment key ck to all the servers in $\mathcal{S}$.

After that, $\mathcal{F}_{\mathsf{V.Offline}}$ generates n_1 random values and n_2 random Beaver triples, and shares and commits them to all MPC servers $S_i \in \mathcal{S}$. If all the MPC servers are corrupted, we let the adversary choose all the random values and Beaver triples; however, if there is an invalid Beaver triple, $\mathcal{F}_{\mathsf{V.Offline}}$ sets the audit flag to $\mathsf{invalid}$. Subsequently, it will be detected by V who sends $(\textsc{Verify}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{V.Offline}}$. In F.6, we provide the realization of $\mathcal{F}_{\mathsf{V.Offline}}$ using CVZK in the $\mathcal{H}$-EUC model.

The online phase. We provide details on the description the description full multi-server VMPC protocol $\Pi_{\mathsf{online}}^{\mathcal{G}_{\mathsf{BB}}, \mathcal{F}_{\mathsf{sc}}, \mathcal{F}_{\mathsf{auth}}, \mathcal{F}_{\mathsf{V.Offline}}}$ presented in Fig. 5.

At the **Initialize** phase, each server $S_i \in \mathcal{S}$ sends $(\textsc{Init}, \mathsf{sid}, 2n, n')$ to $\mathcal{F}_{\mathsf{V.Offline}}$, where n' is the upper-bound of the multiplication gates of the evaluation circuit. When all the servers have sent the init command, $\mathcal{F}_{\mathsf{V.Offline}}$ generates a commitment key ck, $2n$ random values and n' Beaver triples. It then sends the corresponding shares and commitments to the servers $S_i \in \mathcal{S}$. Each server $S_i \in \mathcal{S}$ then posts all the received commitments to $\mathcal{G}_{\mathrm{BB}}$. Next, all servers jointly open $r_{2\ell}$ and $r_{2\ell+1}$ to the client C_ℓ, $\ell \in [n]$ via secure channels. The client C_ℓ checks the validity of $r_{2\ell}$ and $r_{2\ell+1}$ according to their commitments posted on the $\mathcal{G}_{\mathrm{BB}}$, and sets $r_\ell^{(0)} := r_{2\ell}$ and $r_\ell^{(1)} := r_{2\ell+1}$.

At the **Input** phase, the user $U_\ell \in \mathcal{U}$ on input x_ℓ obtains two random numbers $r_\ell^{(0)}$,$r_\ell^{(1)}$ from C_ℓ via $\mathcal{F}_{\mathsf{sc}}$. Then, U_ℓ flips a random bit $b_\ell \leftarrow \{0, 1\}$ and computes $\delta_\ell = x_\ell - r_\ell^{(b_\ell)}$ in $\mathbb{Z}_p$. U_ℓ sends (b_ℓ, δ_ℓ) to C_ℓ via $\mathcal{F}_{\mathsf{sc}}$ and returns the individual audit data $(b_\ell, \delta_\ell, r_\ell^{(1-b_\ell)})$. Finally, C_ℓ posts $(U_\ell, \delta_\ell, b_\ell)$ to $\mathcal{G}_{\mathrm{BB}}$. By flipping b_ℓ, each user U_ℓ, $\ell \in [n]$ contributes one bit of randomness, and all users' bits form the string $(b_1, \ldots, b_n) \in \{0,1\}^n$ that will be provided as input to the verifier coalescence function F presented in Section 4.1. U_ℓ sends (U_ℓ, α_ℓ) to the verifier V via $\mathcal{F}_{\mathsf{auth}}$, where $\alpha_\ell := (b_\ell, \delta_\ell\ r_\ell^{(1-b_\ell)})$.

At the **Compute** phase, each server S_i fetches $\{(U_\ell, \delta_\ell, b_\ell)\}_{\ell \in [n]}$ from $\mathcal{G}_{\mathrm{BB}}$. The server $S_i \in \mathcal{S}$ sets the input as $[\![x_\ell]\!] \leftarrow [\![r_\ell^{(b_\ell)}]\!] + \delta_\ell$ and follows the arithmetic circuit gate by gate, computing over the commitments: (i) for an addition gate for $z = x + y$, S_i sets $[\![z]\!] = [\![x]\!] + [\![y]\!]$; (ii) for multiplication gate $z = x \cdot y$, S_i needs to consume a random Beaver triple $([\![a]\!], [\![b]\!], [\![c]\!])$. S_i opens $[\![x]\!] - [\![a]\!]$ as α and $[\![y]\!] - [\![b]\!]$ as β to $\mathcal{G}_{\mathrm{BB}}$. Once all the other $S_i \in \mathcal{S}$ have opened them to $\mathcal{G}_{\mathrm{BB}}$, S_i then sets $[\![z]\!] = [\![c]\!] + [\![b]\!] \cdot \alpha + [\![a]\!] \cdot \beta + \alpha \cdot \beta$. At the end, $S_i \in \mathcal{S}$ opens the

The functionality $\mathcal{F}_{\mathsf{V.Offline}}$

The functionality operates with the parties $\mathcal{P} = \mathcal{S} \cup \{V\}$. Let set $L_{\mathsf{corr}} \subseteq \mathcal{P}$ contain all corrupted parties. It is parameterized by algorithms $\mathsf{DC.Gen}, \mathsf{DC.Com}$ and variables τ and $\mathcal{J}$. Initially, $\tau = \mathsf{valid}$, $\mathcal{J} = \emptyset$.

- Upon receiving $(\textsc{Init}, \mathsf{sid}, n_1, n_2)$ from an MPC server $S_i \in \mathcal{S}$, it sets $\mathcal{J} = \mathcal{J} \cup \{S_i\}$, and sends a notification message $(\textsc{InitNotify}, \mathsf{sid}, S_i)$ to Sim. If $\mathcal{J} = \mathcal{S}$, it executes the following steps.
  - If $\mathcal{S} \subseteq L_{\mathsf{corr}}$, then it sends $(\mathrm{CK}, \mathsf{sid})$ to Sim. Upon receiving $(\textsc{SetCK}, \mathsf{sid}, \Omega)$ from Sim, it sets $\mathsf{ck} \leftarrow \mathsf{DC.Gen}(1^\lambda; \Omega)$. If $\Omega = \perp$, it sets $\mathsf{ck} \leftarrow \mathsf{DC.Gen}(1^\lambda)$.
  - It sends $(\mathrm{CK}, \mathsf{sid}, \mathsf{ck})$ to all the servers in $\mathcal{S}$.
  - For each server $S_i \in \mathcal{S}$ do:
    - If $S_i \in L_{\mathsf{corr}}$, then it sends $(\textsc{Rand}, \mathsf{sid}, S_i)$ to Sim. Upon receiving $(\textsc{SetRand}, \mathsf{sid}, S_i, (r^*_{i,j}, \rho^*_{i,j})_{j \in [n_1]})$ from Sim, it sets $r_{i,j} = r^*_{i,j}$ and $\rho_{i,j} = \rho^*_{i,j})$ for $j \in [n_1]$.
    - If $S_i \in \mathcal{S} \setminus L_{\mathsf{corr}}$, then it picks random $r_{i,j} \leftarrow \mathbb{Z}_p$ and $\rho_{i,j} \leftarrow (\mathbb{Z}_p)^k$, $j \in [n_1]$.
  - It sends $(\textsc{RandShare}, \mathsf{sid}, (r_{i,j}, \rho_{i,j})_{j \in [n_1]})$ to the honest $S_i \in \mathcal{S} \setminus L_{\mathsf{corr}}$ via private delayed output.
  - It sets $r_j = \sum_{i=1}^k r_{i,j}$, $\rho_j = \sum_{i=1}^k \rho_{i,j}$, and $R_j \leftarrow \mathsf{DC.Com}_{\mathsf{ck}}(r_j; \rho_j)$ for $j \in [n_1]$.
  - It sends $(\textsc{Rand}, \mathsf{sid}, (R_1, \ldots, R_{n_1}))$ to all the servers in $S_i \in \mathcal{S}$.
  - If not all the servers are corrupted, let $S_x \in \mathcal{S} \setminus L_{\mathsf{corr}}$ be an honest server.
  - For each server $S_i \in \mathcal{S}$ do:
    - If $S_i \in L_{\mathsf{corr}}$, then it sends $(\textsc{Triple}, \mathsf{sid}, S_i)$ to Sim. Upon receiving $(\textsc{SetTriple}, \mathsf{sid}, S_i, (a^*_{i,j}, \alpha^*_{i,j}, b^*_{i,j}, \beta^*_{i,j}, c^*_{i,j}, \gamma^*_{i,j})_{j \in [n_2]})$ from Sim, it sets $a_{i,j} = a^*_{i,j}$, $\alpha_{i,j} = \alpha^*_{i,j}$, $b_{i,j} = b^*_{i,j}$, $\beta_{i,j} = \beta^*_{i,j}$, $c_{i,j} = c^*_{i,j}$, and $\gamma_{i,j} = \gamma^*_{i,j}$ for $j \in [n_2]$.
    - If $S_i \in \mathcal{S} \setminus L_{\mathsf{corr}}$ and $i \neq x$, it picks random $a_{i,j}, b_{i,j}, c_{i,j} \leftarrow \mathbb{Z}_p$ and $\alpha_{i,j}, \beta_{i,j}, \gamma_{i,j} \leftarrow (\mathbb{Z}_p)^k$, $j \in [n_2]$.
  - If $\mathcal{S} \subseteq L_{\mathsf{corr}}$, then for $j \in [n_2]$, it computes $a_j = \sum_{i=1}^k a_{i,j}$, $b_j = \sum_{i=1}^k b_{i,j}$, $c_j = \sum_{i=1}^k c_{i,j}$, $\alpha_j = \sum_{i=1}^k \alpha_{i,j}$, $\beta_j = \sum_{i=1}^k \beta_{i,j}$, and $\gamma_j = \sum_{i=1}^k \gamma_{i,j}$.
    If $\exists j \in [n_2] : c_j \neq a_j \cdot b_j$, then it sets $\tau = \mathsf{invalid}$.
  - Otherwise, for S_x, it picks random $a_{i,j}, b_{i,j} \leftarrow \mathbb{Z}_p$ and $\alpha_{i,j}, \beta_{i,j}, \gamma_{i,j} \leftarrow (\mathbb{Z}_p)^k$, $j \in [n_2]$. It computes $a_j = \sum_{i=1}^k a_{i,j}$, $b_j = \sum_{i=1}^k b_{i,j}$, $\alpha_j = \sum_{i=1}^k \alpha_{i,j}$, $\beta_j = \sum_{i=1}^k \beta_{i,j}$, and $\gamma_j = \sum_{i=1}^k \gamma_{i,j}$. It then computes $c_j = a_j \cdot b_j$, $j \in [k]$ and $c_{x,j} = c_j - \sum_{i \in [k] \setminus \{x\}} c_{i,j}$.
  - For each honest $S_i \in \mathcal{S} \setminus L_{\mathsf{corr}}$, it sends $(\textsc{TripleShare}, , \mathsf{sid}, (a_{i,j}, \alpha_{i,j}, b_{i,j}, \beta_{i,j}, c_{i,j}, \gamma_{i,j})_{j \in [n_2]})$ to S_i via private delayed channel.
  - For $j \in [n_2]$, it computes $A_j \leftarrow \mathsf{DC.Com}_{\mathsf{ck}}(a_j; \alpha_j)$, $B_j \leftarrow \mathsf{DC.Com}_{\mathsf{ck}}(b_j; \beta_j)$, and $C_j \leftarrow \mathsf{DC.Com}_{\mathsf{ck}}(c_j; \gamma_j)$. It sends $(\textsc{Triple}, \mathsf{sid}, (A_j, B_j, C_j)_{j \in [n_2]})$ to all $S_i \in \mathcal{S}$.
- Upon receiving $(\textsc{Verify}, \mathsf{sid})$ from V, it returns $(\textsc{Verified}, \mathsf{sid}, \tau)$.

Fig. 4: The functionality $\mathcal{F}_{\mathsf{V.Offline}}$.

output gate(s) of the arithmetic circuit as the final result to $\mathcal{G}_{\mathrm{BB}}$. Furthermore, the servers $S_i \in \mathcal{S}$ jointly open all commitments to all shares $r_{i,\ell}^{(1-b_\ell)}$, $\ell \in [n]$ for auditing.

At the **Verify** phase, upon receiving the users' audit data $(b_\ell, \delta_\ell\ r_\ell^{(1-b_\ell)})$, the verifier V performs the following steps:

(1) Ask all users via $\mathcal{F}_{\mathsf{auth}}$ for their audit data $\langle \alpha_1, \ldots, \alpha_n \rangle$. Upon receiving them parse each $\alpha_\ell = (b_\ell, \delta_\ell, r_\ell^{(1-b_\ell)})$, for $\ell \in [n]$. Now for each ℓ do the following set (i-v) of checks increasing *counter* for each set that successfully passes:
   (i) For $\ell \in [n]$, check b_ℓ and δ_ℓ are consistent with those posted on the BB.
   (ii) Sends $(\textsc{Verify}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{V.Offline}}$ and obtain $(\textsc{Verified}, \mathsf{sid}, \tau)$. Assert that $\tau = \mathsf{valid}$.
   (iii) It verifies that the commitments to all unused random value $r_\ell^{(1-b_\ell)}$, $\ell \in [n]$ are correctly opened.
   (iv) It verifies that opening of all gate commitments are valid.
   (v) It re-computes the arithmetic circuit using the commitments and openings posted on the $\mathcal{G}_{\mathrm{BB}}$ obtaining the evaluation $y = f(x_1, \ldots, x_n)$.

(2) If *counter* $\geq \min\{n, n^{1-1/\gamma}/\log^3(n) + \delta\}$, and set of checks (i-v) verify for ℓ send via $\mathcal{F}_{\mathsf{auth}}$ to U_ℓ $(y, 1)$, else send $(y, 0)$.

Theorem 4. *Let $\Pi_{\mathsf{online}}^{\mathcal{G}_{\mathsf{BB}}, \mathcal{F}_{\mathsf{sc}}, \mathcal{F}_{\mathsf{auth}}, \mathcal{F}_{\mathsf{V.Offline}}}$ be the VMPC scheme in Fig. 5 with n users. Let $\gamma > 1$ be a constant such that $n = \lambda^\gamma$. Let $f : X^n \longrightarrow Y$ be a symmetric function and $R \subseteq \mathsf{Img}[f] \times \mathsf{Img}[f]$ be a δ-spreading relation over $\mathsf{Img}[f]$. The protocol $\Pi_{\mathsf{online}}^{\mathcal{G}_{\mathsf{BB}}, \mathcal{F}_{\mathsf{sc}}, \mathcal{F}_{\mathsf{auth}}, \mathcal{F}_{\mathsf{V.Offline}}}$ $\mathcal{H}$-EUC realizes $\mathcal{F}_{\mathsf{vmpc}}^{f,R}(\mathcal{P})$ in the $\{\mathcal{G}_{\mathsf{BB}}, \mathcal{F}_{\mathsf{sc}}, \mathcal{F}_{\mathsf{auth}}, \mathcal{F}_{\mathsf{V.Offline}}\}$-hybrid model with error $2^{-\delta} + \mathsf{negl}(\lambda)$ under the adaptive DDH assumption, against any PPT environment $\mathcal{Z}$ that statically corrupts at most $\frac{n^{1-\frac{1}{\gamma}}}{\log^3 n}$ users, assuming the underlying CVZK is $(n, \mathsf{negl}(\lambda))$-crowd verifiable complete, $\left(\frac{n^{1-\frac{1}{\gamma}}}{\log^3 n}, \mathsf{negl}(\lambda)\right)$-crowd verifiable sound, and n-crowd verifiable zero-knowledge.*

Proof. (Sketch) Let $\mathcal{A}$ be a real-world PPT adversary that statically corrupts no more than $\frac{n^{1-\frac{1}{\gamma}}}{\log^3 n}$ users. This means that we consider environments that instruct no more than $\frac{n^{1-\frac{1}{\gamma}}}{\log^3 n}$ user corruptions. To prove the theorem, it suffices to construct a simulator Sim such that no non-uniform PPT environment $\mathcal{Z}$ that instruct no more than $\frac{n^{1-\frac{1}{\gamma}}}{\log^3 n}$ user corruptions can distinguish between

(a). the real/hybrid execution, $\mathsf{EXEC}_{\mathcal{A},\mathcal{Z},\mathcal{H}}^{\mathcal{P},\Pi^{\mathcal{G}_{\mathsf{BB}}, \mathcal{F}_{\mathsf{sc}}, \mathcal{F}_{\mathsf{auth}}, \mathcal{F}_{\mathsf{V.Offline}}}}(\lambda)$, in the $\{\mathcal{G}_{\mathrm{BB}}, \mathcal{F}_{\mathsf{sc}}, \mathcal{F}_{\mathsf{auth}}, \mathcal{F}_{\mathsf{V.Offline}}\}$-hybrid world and the corrupted parties are controlled by a dummy adversary $\mathcal{A}$ who simply forwards messages from/to $\mathcal{Z}$, and

(b). the ideal execution, $\mathsf{EXEC}_{\mathsf{Sim},\mathcal{Z},\mathcal{H}}^{\mathcal{F}_{\mathsf{vmpc}}^{f,R}(\mathcal{P}),\mathcal{G}_{\mathsf{BB}}}(\lambda)$, where the parties interact with functionality $\mathcal{F}_{\mathsf{vmpc}}^{f,R}(\mathcal{P})$ in the $\mathcal{G}_{\mathrm{BB}}$-hybrid model and corrupted parties are controlled by the simulator Sim.

Moreover, there is a super-polynomial helper $\mathcal{H}$ that exists in both ideal and real world.

Initialize.

1. Upon receiving $(\textsc{Start}, \mathsf{sid})$ from the environment $\mathcal{Z}$, the server $S_i \in \mathcal{S}$ sends $(\textsc{Init}, \mathsf{sid}, 2n, n')$ to $\mathcal{F}_{\mathsf{V.Offline}}$, where n' is the number of multiplication gates.
2. Upon receiving $(\mathrm{CK}, \mathsf{sid}, \mathsf{ck})$, $(\textsc{RandShare}, \mathsf{sid}, (r_{i,j}, \rho_{i,j})_{j\in[2n]})$, $(\textsc{Rand}, \mathsf{sid}, (R_1, \ldots, R_{2n}))$, $(\textsc{TripleShare}, , \mathsf{sid}, (a_{i,j}, \alpha_{i,j}, b_{i,j}, \beta_{i,j}, c_{i,j}, \gamma_{i,j})_{j\in[n']})$ and $(\textsc{Triple}, \mathsf{sid}, (A_j, B_j, C_j)_{j\in[n']})$ from $\mathcal{F}_{\mathsf{V.Offline}}$, the server $S_i \in \mathcal{S}$
   - Posts $(\mathsf{ck}, (R_j)_{j\in[2n]}, (A_j, B_j, C_j)_{j\in[n']})$ to the $\mathcal{G}_{\text{BB}}$.
   - For $\ell \in [n]$, sends $(r_{i,2\ell}, \rho_{i,2\ell})$ and $(r_{i,2\ell+1}, \rho_{i,2\ell+1})$ to C_ℓ.
3. Upon receiving $(r_{i,2\ell}, \rho_{i,2\ell})$ and $(i, r_{2\ell+1}, \rho_{i,2\ell+1})$ from all the servers $S_i \in \mathcal{S}$, the client C_ℓ sets $r_\ell^{(0)} = \sum_{j=1}^k r_{j,2\ell}$, $\rho_\ell^{(0)} = \sum_{j=1}^k \rho_{j,2\ell}$, $r_\ell^{(1)} = \sum_{j=1}^k r_{j,2\ell+1}$, and $\rho_\ell^{(1)} = \sum_{j=1}^k \rho_{j,2\ell+1}$. It then fetches ck, $R_{2\ell}$ and $R_{2\ell+1}$ from the BB and checks if $R_{2\ell} = \mathsf{DC.Com}_{\mathsf{ck}}(r_\ell^{(0)}; \rho_\ell^{(0)})$ and $R_{2\ell+1} = \mathsf{DC.Com}_{\mathsf{ck}}(r_\ell^{(1)}; \rho_\ell^{(1)})$.

Input.

4. Upon receiving $(\textsc{Cast}, \mathsf{sid}, x_\ell)$ from the environment $\mathcal{Z}$, the user U_ℓ fetches $(r_\ell^{(0)}, r_\ell^{(1)})$ from C_ℓ via $\mathcal{F}_{\mathsf{sc}}$. It then picks $b_\ell \leftarrow \{0,1\}$ and computes $\delta_\ell = x_\ell - r_\ell^{(b_\ell)}$. Next, it sends (b_ℓ, δ_ℓ) to C_ℓ via $\mathcal{F}_{\mathsf{sc}}$.
5. Upon receiving (b_ℓ, δ_ℓ) from U_ℓ, C_ℓ posts $(U_\ell, b_\ell, \delta_\ell)$ to $\mathcal{G}_{\text{BB}}$ and stores (U_ℓ, α_ℓ).

Compute.

6. Upon receiving $(\textsc{Compute}, \mathsf{sid})$ from the environment $\mathcal{Z}$, the server S_i:
   - Fetches the posted $(U_\ell, b_\ell, \delta_\ell)_{\ell\in[n]}$ from $\mathcal{G}_{\text{BB}}$ (via a $\textsc{Read}$ requests), and computes $[\![x_\ell]\!] \leftarrow [\![r_\ell^{(b_\ell)}]\!] + \delta_\ell$. For $\ell \in [n]$, it then opens $R_\ell^{(1-b_\ell)}$ to the $\mathcal{G}_{\text{BB}}$.
   - Uses $([\![x_\ell]\!])_{\ell\in[n]}$ as input commitments and follows the arithmetic circuit f gate by gate, computing over the commitments:
     (i) For an addition gate for $z = x + y$, it sets $[\![z]\!] = [\![x]\!] + [\![y]\!]$.
     (ii) For a multiplication gate $z = x \cdot y$, it consumes a random Beaver triple $([\![a]\!], [\![b]\!], [\![c]\!])$. S_i opens $[\![x]\!] - [\![a]\!]$ as α and $[\![y]\!] - [\![b]\!]$ as β to $\mathcal{G}_{\text{BB}}$. Once all the other $S_i \in \mathcal{S}$ have opened them to $\mathcal{G}_{\text{BB}}$, S_i then sets $[\![z]\!] = [\![c]\!] + [\![b]\!] \cdot \alpha + [\![a]\!] \cdot \beta + \alpha \cdot \beta$.
     (iii) For an output gate, it opens the commitment to $\mathcal{G}_{\text{BB}}$.

Verify.

7. Upon receiving $(\textsc{Verify}, \mathsf{sid})$ from the environment $\mathcal{Z}$, the verifier V
   (1) Ask all users via $\mathcal{F}_{\mathsf{auth}}$ for their receipts $\langle \alpha_1, \ldots, \alpha_n \rangle$. Upon receiving them parse each $\alpha_\ell = (b_\ell, \delta_\ell, r_\ell^{(1-b_\ell)})$, for $\ell \in [n]$. Now for each ℓ do the following set (i-v) of checks increasing *counter* for each set that successfully passes:
     (i) For $\ell \in [n]$, check b_ℓ and δ_ℓ are consistent with those posted on the BB.
     (ii) Sends $(\textsc{Verify}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{V.Offline}}$ and obtain $(\textsc{Verified}, \mathsf{sid}, \tau = \mathsf{valid})$.
     (iii) It verifies that the commitments to all unused random value $r_\ell^{(1-b_\ell)}$, $\ell \in [n]$ are correctly opened.
     (iv) It verifies that opening of all gate commitments are valid.
     (v) It re-computes the arithmetic circuit using the commitments and openings posted on the $\mathcal{G}_{\text{BB}}$ obtaining the evaluation $y = f(x_1, \ldots, x_n)$.
   (2) If *counter* $\geq \min\{n, n^{1-1/\gamma}/\log^3(n) + \delta\}$, and set of checks (i-v) verify for ℓ send via $\mathcal{F}_{\mathsf{auth}}$ to U_ℓ $(y, 1)$, else send $(y, 0)$.

Fig. 5: VMPC protocol $\mathit{\Pi}_{\mathsf{online}}^{\mathcal{G}_{\mathsf{BB}}, \mathcal{F}_{\mathsf{sc}}, \mathcal{F}_{\mathsf{auth}}, \mathcal{F}_{\mathsf{V.Offline}}}$.

As a first step, we apply Theorem 1 and Theorem 2 given that the number of corrupted verifiers (users) is no more than $\frac{n^{1-\frac{1}{\gamma}}}{\log^3 n}$ to deduce the security error of the CVZK proofs. When $n = \lambda^\gamma$ and the adversary can corrupt up to $\frac{n^{1-\frac{1}{\gamma}}}{\log^3 n}$ users, the underlying coalescence construction of the CVZK construction is a $\left(\frac{\lambda}{\log^2 \lambda}, \log \lambda, \frac{n^{1-\frac{1}{\gamma}}}{\log^3 n}\right)$-coalescence function. Therefore, we get that there is a negligible function $\epsilon(\cdot)$ such that the following hold:

(1) The CVZK proofs satisfy (perfect) $(n, \epsilon(\lambda))$-crowd verifiable completeness.

(2) For every $t_2 \leq \frac{n^{1-\frac{1}{\gamma}}}{\log^3 n}$, the CVZK proofs satisfy $(t_2, \epsilon(\lambda))$-crowd verifiable soundness.

(3) The CVZK proofs satisfy n-crowd-verifiable zero-knowledge in the $\mathcal{H}$-helper model.

Let $\mathcal{U}_{\mathsf{corr}}$, $\mathcal{C}_{\mathsf{corr}}$ and $\mathcal{S}_{\mathsf{corr}}$ be the corrupted set of users, clients, and servers, respectively, where we have that $|\mathcal{U}_{\mathsf{corr}}| \leq \frac{n^{1-\frac{1}{\gamma}}}{\log^3 n}$. We consider the following two cases.

Case 1: $0 \leq |\mathcal{S}_{\mathsf{corr}}| < k$ (i.e., there is at least one honest server).

Simulation. The simulator Sim simulates $\mathcal{F}_{\mathsf{V.Offline}}$, so Sim can learn all the shares of generated random coins $\{r_j\}_{j\in[2n]}$ and Beaver triples $\{(a_j, b_j, c_j)\}_{j\in[n']}$, because all the parties receive shares from $\mathcal{F}_{\mathsf{V.Offline}}$. The proof is straightforward, for malicious user $U_\ell \in \mathcal{U}_{\mathsf{corr}}$, the simulator Sim can extract their inputs by computing $\delta_\ell + r_\ell$, where r_ℓ is the used random value and δ_ℓ is the posted difference on the $\mathcal{G}_{\mathsf{BB}}$. The simulator then submit the malicious users' input to $\mathcal{F}^{f,R}_{\mathsf{vmpc}}(\mathcal{P})$; for honest user $U_\ell \in \mathcal{U} \setminus \mathcal{U}_{\mathsf{corr}}$, the simulator uses 0 (or any other fixed element in $\mathbb{Z}_p$) as their input and follows the protocol description as if the user U_ℓ receives command $(\textsc{Cast}, \mathsf{sid}, 0)$ from the environment $\mathcal{Z}$. At the end of the computing phase, when simulator Sim obtains the outcome from $\mathcal{F}^{f,R}_{\mathsf{vmpc}}(\mathcal{P})$, it needs to fake the final opening of the commitment by simulating the CVZK opening. The step needs $\mathcal{H}$'s help to enable the CVZK simulation.

Indistinguishability. Since there is at least one honest server $S_i \in \mathcal{S}$, the soundness of all the ZK proofs is ensured by the randomness generated by the honest server, regardless the number of corrupted users. The indistinguishability between the view of $\mathcal{Z}$ and $\mathcal{A}$ in the real execution $\mathsf{EXEC}^{\mathcal{P},\Pi^{\mathcal{G}_{\mathsf{BB}},\mathcal{F}_{\mathsf{sc}},\mathcal{F}_{\mathsf{auth}},\mathcal{F}_{\mathsf{V.Offline}}}}_{\mathcal{A},\mathcal{Z},\mathcal{H}}(\lambda)$ and the ideal execution $\mathsf{EXEC}^{\mathcal{F}^{f,R}_{\mathsf{vmpc}}(\mathcal{P}),\mathcal{G}_{\mathsf{BB}}}_{\mathsf{Sim},\mathcal{Z},\mathcal{H}}(\lambda)$, derives by the security of the CVZK proofs mentioned above in the $\mathcal{H}$-EUC model. Finally, we consider the output/return distribution of the command $(\textsc{Verify}, \mathsf{sid})$ of the (honest) auditor V between $\mathsf{EXEC}^{\mathcal{P},\Pi^{\mathcal{G}_{\mathsf{BB}},\mathcal{F}_{\mathsf{sc}},\mathcal{F}_{\mathsf{auth}},\mathcal{F}_{\mathsf{V.Offline}}}}_{\mathcal{A},\mathcal{Z},\mathcal{H}}(\lambda)$ and $\mathsf{EXEC}^{\mathcal{F}^{f,R}_{\mathsf{vmpc}}(\mathcal{P}),\mathcal{G}_{\mathsf{BB}}}_{\mathsf{Sim},\mathcal{Z},\mathcal{H}}(\lambda)$.

As discussed in the single-server case (cf. 7.2), the adversary $\mathcal{A}$ may perform so-called *Client attacks*. Namely, the adversary $\mathcal{A}$ can corrupt the client C_ℓ of U_ℓ and provide U_ℓ with a pair of random values $(\hat{r}^{(0)}_\ell, \hat{r}^{(1)}_\ell)$, where one component $\hat{r}^{(b^*)}_\ell$ is different than $r^{(b^*)}_\ell$ in the pair $(r^{(0)}_\ell, r^{(1)}_\ell)$ committed in $\mathcal{G}_{\mathsf{BB}}$. Hence, if $\mathcal{A}^*$ guesses the coin of U_ℓ correctly (i.e. $b^* = b_\ell$), then it can perform the VMPC execution by replacing U_ℓ's input x_ℓ with input $x^*_\ell = x_\ell + \big(\hat{r}^{(b^*)}_\ell - r^{(b^*)}_\ell\big)$ without being detected. Given that U_ℓ flips a fair coin, this attack has $1/2$ success probability. When $f : X^n \longrightarrow Y$ is a symmetric function and $R \subseteq \mathsf{Img}[f] \times \mathsf{Img}[f]$ is a δ-spreading relation, the probability that $(y, \tilde{y}) \notin R$ but it is not detected by $\Pi^{\mathcal{G}_{\mathsf{BB}},\mathcal{F}_{\mathsf{sc}},\mathcal{F}_{\mathsf{auth}},\mathcal{F}_{\mathsf{V.Offline}}}$ is bounded by

$$P = (1/2)^\delta + \mathsf{negl}(\lambda) \ .$$

Therefore, when $(y, \tilde{y}) \notin R$, the (honest) auditor V will output $(\textsc{Result}, \mathsf{sid}, \tilde{y}, 1)$ to U_ℓ in the protocol $\Pi^{\mathcal{G}_{\mathsf{BB}},\mathcal{F}_{\mathsf{sc}},\mathcal{F}_{\mathsf{auth}},\mathcal{F}_{\mathsf{V.Offline}}}$ execution. The same occurs in $\mathcal{F}^{f,R}_{\mathsf{vmpc}}(\mathcal{P})$ according to the condition (2) in the verification checking condition of Fig. 3. On the other hand, if $\mathcal{A}$ performs less than δ client attacks, then the actual input $\mathbf{x}$ and the adversarial one $\mathbf{x}'$ are δ-close w.r.t. $\mathsf{Dcr}_n(\cdot,\cdot)$. Since the relation R is δ-spreading, we have that $R\big(f(\mathbf{x}), f(\mathbf{x}')\big)$ holds. So, when the simulator plays the role of the verifier accepts, it sends an accept message ($\tilde{v} = 1$) for every honest user to $\mathcal{F}^{f,R}_{\mathsf{vmpc}}(\mathcal{P})$ which in turn will also accept. Besides, $\mathcal{F}^{f,R}_{\mathsf{vmpc}}(\mathcal{P})$ will reject whenever the simulator sends a reject message, hence, indistinguishability is again preserved.

Case 2: $0 \le |\mathcal{U}_{\mathsf{corr}}| \le \frac{n^{1-\frac{1}{\gamma}}}{\log^3 n} \ \wedge \ |\mathcal{S}_{\mathsf{corr}}| = k$ (full server corruption).

The case of full server corruption, has similarities with the argumentation of the single-server example (cf. 7.2), as full corruption reduces to a single malicious server setting. Thus, only end-to-end verifiability is expected to be preserved.

No privacy is guaranteed with all the servers are corrupted. All the users' inputs are leaked by the $\mathcal{F}^{f,R}_{\mathsf{vmpc}}(\mathcal{P})$, the simulator Sim can simply simulate the view by following the protocol description with the leaked inputs. Since for any $t_2 \le \frac{n^{1-\frac{1}{\gamma}}}{\log^3 n}$, the underlying CVZK proofs satisfy $(t_2, \epsilon(\lambda))$-crowd verifiable soundness and t_2-crowd verifiable validity, when $|\mathcal{U}_{\mathsf{corr}}| \le \frac{n^{1-\frac{1}{\gamma}}}{\log^3 n}$, the soundness of all the ZK proofs are ensured (up to $\mathsf{negl}(\lambda)$ error) even when all the servers are corrupted. Moreover, as there is no simulated message, the view of $\mathcal{Z}$ and $\mathcal{A}$ in the real execution $\mathsf{EXEC}^{\mathcal{P},\Pi^{\mathcal{G}_{\mathsf{BB}},\mathcal{F}_{\mathsf{sc}},\mathcal{F}_{\mathsf{auth}},\mathcal{F}_{\mathsf{V.Offline}}}}_{\mathcal{A},\mathcal{Z},\mathcal{H}}(\lambda)$ is identical to the ideal execution $\mathsf{EXEC}^{\mathcal{F}^{f,R}_{\mathsf{vmpc}}(\mathcal{P}),\mathcal{G}_{\mathsf{BB}}}_{\mathsf{Sim},\mathcal{Z},\mathcal{H}}(\lambda)$. With regards to the end-to-end verifiability, the output/return distribution of the command $(\textsc{Verify}, \mathsf{sid})$ of the (honest) auditor V between $\mathsf{EXEC}^{\mathcal{P},\Pi^{\mathcal{G}_{\mathsf{BB}},\mathcal{F}_{\mathsf{sc}},\mathcal{F}_{\mathsf{auth}},\mathcal{F}_{\mathsf{V.Offline}}}}_{\mathcal{A},\mathcal{Z},\mathcal{H}}(\lambda)$ and $\mathsf{EXEC}^{\mathcal{F}^{f,R}_{\mathsf{vmpc}}(\mathcal{P}),\mathcal{G}_{\mathsf{BB}}}_{\mathsf{Sim},\mathcal{Z},\mathcal{H}}(\lambda)$ are indistinguishable when $f : X^n \longrightarrow Y$ is a symmetric function and $R \subseteq \mathsf{Img}[f] \times \mathsf{Img}[f]$ is a δ-spreading relation. The proof is similar to that of **Case 1**.

We note that if $\mathcal{A}$ additionally corrupts the verifier V, we can construct a simple simulator that engages with $\mathcal{A}$ plays the role of honest users and simply forwards the malicious response of V to $\mathcal{F}^{f,R}_{\mathsf{vmpc}}(\mathcal{P})$ along with the adversarial tally y'.

Therefore, $\Pi^{\mathcal{G}_{\mathsf{BB}},\mathcal{F}_{\mathsf{sc}},\mathcal{F}_{\mathsf{auth}},\mathcal{F}_{\mathsf{V.Offline}}}_{\mathsf{online}}$ $\mathcal{H}$-EUC realizes $\mathcal{F}^{f,R}_{\mathsf{vmpc}}(\mathcal{P})$ with error $2^{-\delta} + \mathsf{negl}(\lambda)$ in the $\{\mathcal{G}_{\mathsf{BB}}, \mathcal{F}_{\mathsf{sc}}, \mathcal{F}_{\mathsf{auth}}, \mathcal{F}_{\mathsf{V.Offline}}\}$-hybrid model, against any PPT environment $\mathcal{Z}$ that statically corrupts no more than $\frac{n^{1-\frac{1}{\gamma}}}{\log^3 n}$ users. □

Remark: When $\delta = \omega(\log \lambda)$, then $\Pi_{\mathsf{online}}^{\mathcal{G}_{\mathsf{BB}}, \mathcal{F}_{\mathsf{sc}}, \mathcal{F}_{\mathsf{auth}}, \mathcal{F}_{\mathsf{V.Offline}}}$ $\mathcal{H}$-EUC realizes $\mathcal{F}_{\mathsf{vmpc}}^{f,R}(\mathcal{P})$.

Efficiency. The overall computational and communicational cost is $O(\lambda \cdot (n + k) \cdot |C|)$, where $|C|$ stands for the number of multiplicative gates of the circuit, n is the number of users, and k is the number of servers. The concrete efficiency of VMPC largely depends on the efficiency of the instantiation of the underlying fully input-delayed Σ-protocol used in CVZK.

8 Applications of VMPC

Examples of interesting VMPC application scenarios may refer to e-voting, as well as any type of privacy-preserving data processing where for transparency reasons, it is important to provide evidence of the integrity of the outcome, e.g., demographic statistics or financial analysis. In our modeling, the most appealing cases - in terms of usability by a user with “human level” limitations - are the ones where the error is small for the lowest possible entropy, e.g. users contribute only 1 bit. Hence, for simplicity we set $\kappa = 1$. Following the reasoning in Section 6 and by Theorem 3, when $\kappa = 1$, a VMPC application can be feasible when it is w.r.t. to δ-spreading relations and with an error expected to be $\mathsf{negl}(\delta)$ (ignoring the $\mathsf{negl}(\lambda)$ cryptographic error). In general, we can calibrate the security error by designing VMPC schemes that support sufficiently large values of κ. We present a selection of interesting VMPC applications below.

e-Voting. The security analysis of several e-voting systems (e.g. [48, 44, 23]) is based on the claim that “assuming cryptographic security, by attacking one voter you change one vote, thus you add at most one to the total tally deviation”. This claim can be seen as a special case of VMPC security for an evaluation (tally) function which is 1-Lipschitz and tally deviation is naturally captured by R_δ defined in Eq. (2). Thus, if the voters contribute min entropy of 1 bit, then we expect that e-voting security holds with error $\mathsf{negl}(\delta)$.

Privacy-preserving statistics. Let $X = [a, b]$ be a range of integer values, $Y = [na, nb]$ and $f := \frac{\sum_{\ell=1}^{n} x_\ell}{n}$ be the average of all users' inputs. E.g., $[a, b]$ could be the number of unemployed adults or dependent members in a family, the range of the employees' salary in a company, or the household power consumption in a city measured by smart meters. If we set d_Y to the absolute value $|\cdot|$, then f is a $\frac{b-a}{n}$-Lipschitz function for Dcr_n and $|\cdot|$, so for user min entropy of 1 bit, we expect that (f, R_δ) can be realized with error $\mathsf{negl}(\frac{\delta n}{b-a})$. This also generalizes to other aggregate statistics such as calculating higher moments over the data set.

Privacy-preserving processing of multidimensional data (profile matching). A useful generalization of the privacy-preserving statistics case is when performing processing on multidimensional data collected from multiple sources. A simple two-dimensional example illustrating this follows. Let X_1, X_2 be two domains of attributes and $X := X_1 \times X_2$, i.e. each input x_ℓ is an attribute pair $(x_{\ell,1}, x_{\ell,2})$. Let $Y = [n]$, P_1, P_2 be predicates over X_1, X_2 respectively and let $f := \sum_{\ell=1}^{n} P_1(x_{\ell,1}) \cdot P_2(x_{\ell,2})$ be the function that counts the number of inputs that satisfy both P_1, P_2. E.g., X_1 could be the set of dates and X_2 be the locations, fragmented in area units. Then, f counts the number of people that are in a specific place and have their birthday. If we set d_Y to $|\cdot|$, then f is a 1-Lipschitz function for Dcr_n and $|\cdot|$. Like the previous example, we expect that (f, R_δ) can be realized with error $\mathsf{negl}(\delta)$.

Supervised learning of (binary) classifiers. In many use cases, functions that operate as classifiers are being "trained" via a machine learning algorithm (e.g. Perceptron) on input a vector of training data. Here, we view the users' inputs as training data that are vectors of dimension m, i.e. $x_\ell = (x_{\ell,1}, \dots, x_{\ell,m}) \in [a_1, b_1] \times \cdots \times [a_m, b_m]$, where $[a_i, b_i]$, $i \in [m]$ are intervals. The evaluation function f outputs a hyperplane $HP(\mathbf{x}) := \{\mathbf{w} \cdot \mathbf{z} \mid \mathbf{z} \in \mathbb{R}^m\}$ that defines the decision's 0/1 output. If the adversary changes $\mathbf{x}$ with some $\mathbf{x}'$ s.t. $\mathsf{Dcr}_n(\mathbf{x}, \mathbf{x}') \leq \delta$, then the adversarially computed hyperplane $HP(\mathbf{x}') := \{\mathbf{w}' \cdot \mathbf{z} \mid \mathbf{z} \in \mathbb{R}^m\}$ must be close to $HP(\mathbf{x})$, otherwise the attack is detected. This could be expressed by having $\mathbf{w}, \mathbf{w}'$ be δ close w.r.t. the Euclidean distance. Assume now that for a set of new data points $\mathbf{z}_1, \dots, \mathbf{z}_t$ we set the relation as "$R\big(HP(\mathbf{x}), HP(\mathbf{x}')\big) \Leftrightarrow \forall j \in [t]$ the classifier makes the same decision for $\mathbf{z}_j$". Then, clearly R is a spreading relation w.r.t. to f, suggesting that the functionality of calculating classifier is resilient against attacks on less than δ of the training data.

References

1. M. Ajtai and N. Linial. The influence of large coalitions. *Combinatorica*, 13(2):129–145, 1993.
2. J. Alwen, R. Ostrovsky, H. Zhou, and V. Zikas. Incoercible multi-party computation and universally composable receipt-free voting. In *CRYPTO*, 2015.
3. B. Barak, R. Canetti, Y. Lindell, R. Pass, and T. Rabin. Secure computation without authentication. In *CRYPTO*, 2005.
4. C. Baum, I. Damgård, and C. Orlandi. Publicly auditable secure multi-party computation. In *SCN*, 2014.
5. C. Baum, I. Damgård, and C. Orlandi. Publicly auditable secure multi-party computation. In M. Abdalla and R. D. Prisco, editors, *SCN 14*, volume 8642 of *LNCS*, pages 175–196. Springer, Heidelberg, Sept. 2014.
6. D. Beaver. Efficient multiparty protocols using circuit randomization. In *CRYPTO*, 1991.
7. D. Beaver. Commodity-based cryptography (extended abstract). In *29th ACM STOC*, pages 446–455. ACM Press, May 1997.
8. M. Bellare, G. Fuchsbauer, and A. Scafuro. NIZKs with an untrusted CRS: security in the face of parameter subversion. In *ASIACRYPT*, 2016.
9. M. Ben-Or and N. Linial. Collective coin flipping, robust voting schemes and minima of banzhaf values. In *FOCS*, 1985.
10. J. Benaloh. Simple verifiable elections. USENIX EVT. USENIX Association, 2006.
11. J. Benaloh. Ballot casting assurance via voter-initiated poll station auditing. In *EVT*, 2007.
12. R. Bendlin, I. Damgård, C. Orlandi, and S. Zakarias. Semi-homomorphic encryption and multiparty computation. In *EUROCRYPT*, 2011.

13. P. Bogetoft, D. L. Christensen, I. Damgård, M. Geisler, T. P. Jakobsen, M. Krøigaard, J. D. Nielsen, J. B. Nielsen, K. Nielsen, J. Pagter, M. I. Schwartzbach, and T. Toft. Secure multiparty computation goes live. In *FC*, 2009.
14. R. Bost, R. A. Popa, S. Tu, and S. Goldwasser. Machine learning classification over encrypted data. In *NDSS*, 2015.
15. M. Burmester and Y. Desmedt. Broadcast interactive proofs (extended abstract). In *EUROCRYPT*, 1991.
16. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, 2001.
17. R. Canetti, Y. Dodis, R. Pass, and S. Walfish. Universally composable security with global setup. In *TCC*, 2007.
18. R. Canetti, H. Lin, and R. Pass. Adaptive hardness and composable security in the plain model from standard assumptions. In *FOCS*, 2010.
19. E. Chattopadhyay and D. Zuckerman. Explicit two-source extractors and resilient functions. In *STOC*, 2016.
20. D. Chaum. Secret-ballot receipts: True voter-verifiable elections. *IEEE S&P*, 2004.
21. M. Ciampi, G. Persiano, A. Scafuro, L. Siniscalchi, and I. Visconti. Improved or-composition of sigma-protocols. In *TCC*, 2016.
22. M. Ciampi, G. Persiano, A. Scafuro, L. Siniscalchi, and I. Visconti. Online/offline OR composition of sigma protocols. In *EUROCRYPT*, 2016.
23. V. Cortier, D. Galindo, R. Küsters, J. Mueller, and T. Truderung. SoK: Verifiability notions for e-voting protocols. In *IEEE Security & Privacy*, 2016.
24. R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *CRYPTO*, 1994.
25. R. Cramer, R. Gennaro, and B. Schoenmakers. A secure and optimally efficient multi-authority election scheme. In *EUROCRYPT*, 1997.
26. I. Damgård, K. Damgård, K. Nielsen, P. S. Nordholt, and T. Toft. Confidential benchmarking based on multiparty computation. In *FC*, 2016.
27. I. Damgård, Y. Ishai, M. Krøigaard, J. B. Nielsen, and A. D. Smith. Scalable multiparty computation with nearly optimal work and resilience. In *CRYPTO*, 2008.
28. I. Damgård, M. Keller, E. Larraia, V. Pastro, P. Scholl, and N. P. Smart. Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits. In *ESORICS*, 2013.
29. I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO*, 2012.
30. Y. Dodis, T. Ristenpart, and S. P. Vadhan. Randomness condensers for efficiently samplable, seed-dependent sources. In *TCC*, 2012.
31. C. Ellison. Ceremony design and analysis. IACR ePrint, Report 2007/399, 2007.
32. U. Feige, J. Kilian, and M. Naor. A minimal model for secure computation (extended abstract). In *26th ACM STOC*, pages 554–563. ACM Press, May 1994.
33. N. Fleischhacker, V. Goyal, and A. Jain. On the existence of three round zero-knowledge proofs. In *EUROCRYPT*, 2018.
34. N. Gilboa. Two party RSA key generation. In *CRYPTO*, 1999.
35. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *STOC*, 1987.
36. S. Halevi, Y. Lindell, and B. Pinkas. Secure computation on the web: Computing without simultaneous interaction. In *CRYPTO*, 2011.
37. C. Hazay and M. Venkitasubramaniam. On the power of secure two-party computation. In *CRYPTO*, 2016.

38. Y. Ishai, E. Kushilevitz, and A. Paskin. Secure multiparty computation with minimal interaction. In *CRYPTO*, 2010.
39. J. Kahn, G. Kalai, and N. Linial. The influence of variables on boolean functions (extended abstract). In *FOCS*, 1988.
40. Y. T. Kalai, G. N. Rothblum, and R. D. Rothblum. From obfuscation to the security of Fiat-Shamir for proofs. In *CRYPTO*, 2017.
41. S. Kamara, P. Mohassel, and B. Riva. Salus: a system for server-aided secure function evaluation. In *ACM-CCS*, 2012.
42. M. Keller, E. Orsini, and P. Scholl. MASCOT: faster malicious arithmetic secure computation with oblivious transfer. In *CCS*, 2016.
43. A. Kiayias, T. Zacharias, and B. Zhang. DEMOS-2: scalable E2E verifiable elections without random oracles. In *ACM-CCS*, 2015.
44. A. Kiayias, T. Zacharias, and B. Zhang. End-to-end verifiable elections in the standard model. In *EUROCRYPT*, 2015.
45. A. Kiayias, T. Zacharias, and B. Zhang. Ceremonies for end-to-end verifiable elections. In *PKC*, 2017.
46. B. Kreuter, A. Shelat, and C. Shen. Billion-gate secure computation with malicious adversaries. In *USENIX*, 2012.
47. R. Küsters, T. Truderung, and A. Vogt. Accountability: definition and relationship to verifiability. In *ACM-CCS*, 2010.
48. R. Küsters, T. Truderung, and A. Vogt. Clash attacks on the verifiability of e-voting systems. In *IEEE Security & Privacy*, 2012.
49. D. Lapidot and A. Shamir. Publicly verifiable non-interactive zero-knowledge proofs. In *CRYPTO*, 1990.
50. M. Lepinski, S. Micali, and A. Shelat. Fair-zero knowledge. In *TCC*, 2005.
51. X. Li. Improved two-source extractors, and affine extractors for polylogarithmic entropy. In *FOCS*, 2016.
52. Y. Lindell. An efficient transform from sigma protocols to NIZK with a CRS and non-programmable random oracle. Cryptology ePrint Archive, Report 2014/710, 2014. `http://eprint.iacr.org/2014/710`.
53. Y. Lindell and B. Pinkas. Secure multiparty computation for privacy-preserving data mining. *IACR ePrint 2008/197*, 2008.
54. A. López-Alt, E. Tromer, and V. Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *STOC*, 2012.
55. R. Meka. Explicit resilient functions matching ajtai-linial. In *SODA*, 2017.
56. C. A. Neff. Practical high certainty intent verification for encrypted votes. Votehere, Inc. whitepaper, 2004.
57. J. B. Nielsen, P. S. Nordholt, C. Orlandi, and S. S. Burra. A new approach to practical active-secure two-party computation. In *CRYPTO*, 2012.
58. O. Pandey, R. Pass, and V. Vaikuntanathan. Adaptive one-way functions and applications. In *CRYPTO*, 2008.
59. R. Pass. Simulation in quasi-polynomial time, and its application to protocol composition. In *EUROCRYPT*, 2003.
60. B. Pinkas, T. Schneider, N. P. Smart, and S. C. Williams. Secure two-party computation is practical. In *ASIACRYPT*, 2009.
61. B. Schoenmakers and M. Veeningen. Universally verifiable multiparty computation from threshold homomorphic cryptosystems. In *ACNS*, pages 3–22, 2015.
62. A. C.-C. Yao. Protocols for secure computations (extended abstract). In *23rd FOCS*, pages 160–164. IEEE Computer Society Press, Nov. 1982.
63. A. C.-C. Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, Oct. 1986.

A Supplementary Preliminaries

A.1 Σ-protocols

Definition 8. *Let $\mathcal{L}$ be a language in* **NP** *and $R_{\mathcal{L}}$ be a witness relation for the language $\mathcal{L}$. A protocol $\Sigma.\Pi = (\Sigma.P, \Sigma.V)$ (where $\Sigma.P$ consists of algorithms $(\Sigma.\mathsf{Prv}_1, \Sigma.\mathsf{Prv}_2)$) is said to be a* Σ-protocol *if there is a negligible function $\delta(\cdot)$ such that the following properties hold:*

δ-Completeness: *For every $x \in \mathcal{L} \cap \{0,1\}^{\mathrm{poly}(\lambda)}$ and $w \in R_{\mathcal{L}}(x)$,*

$$\Pr[(a, \mathsf{st}_P) \leftarrow \Sigma.\mathsf{Prv}_1(x, w); e \leftarrow \Sigma.V(x, a); z \leftarrow \Sigma.\mathsf{Prv}_2(x, w, \mathsf{st}_P, e) : \\ \Sigma.\mathsf{Verify}(x, a, e, z) = 0] \leq \delta(\lambda).$$

Special Soundness: *There exists a PPT extractor $\Sigma.\mathsf{Ext}$, such that for every pair of valid transcripts (x, a, e, z) and (x, a, e', z') with $e \neq e'$, i.e. $\Sigma.\mathsf{Verify}(x, a, e, z) = 1$ and $\Sigma.\mathsf{Verify}(x, a, e', z') = 1$, the extractor $\Sigma.\mathsf{Ext}$ can efficiently compute w such that $(x, w) \in R_{\mathcal{L}}$.*

sHVZK: *there exists a PPT simulator $\Sigma.\mathsf{Sim}$ such that for every $x \in \mathcal{L} \cap \{0,1\}^{\mathrm{poly}(\lambda)}$, $w \in R_{\mathcal{L}}(x)$ and $e \in \{0,1\}^{\lambda}$, the following distributions are indistinguishable:*

- $\Sigma.\mathsf{Sim}(x, e)$
- $\{(a, \mathsf{st}_P) \leftarrow \Sigma.\mathsf{Prv}_1(x, w); z \leftarrow \Sigma.\mathsf{Prv}_2(x, w, \mathsf{st}_P, e) : (x, a, e, z)\}$.

A.2 Related Work in Resilient Functions

Ben-Or and Linial showed that the iterative-majority function is $(\varepsilon n^{0.63}, \varepsilon)$-resilient for any $\varepsilon = n^{-\Omega(1)}$ [9]. Further, Ajtai and Linial showed the existence of $(\Omega(n/\log^2 n), 1/3)$-resilient functions [1], but their construction is probabilistic and its de-randomized variant takes $n^{O(n^2)}$ running time. Chattopadhyay and Zuckerman [19] proposed an explicit construction of a monotone, almost balanced $(n^{1-\delta}, n^{-\Omega(1)})$-resilient function, for any $\delta > 0$. Later, Li [51] extended their one-bit construction to $f : \{0,1\}^n \mapsto \{0,1\}^m$ that is still $(n^{1-\delta}, n^{-\Omega(1)})$-resilient. Recently, Meka [55] gave an explicit resilient function for $q = O(n/\log^2 n)$, but the function is a slightly biased. Moreover, the function is so-called $(\Theta(\log^2 n/n))$-*strongly resilient*, i.e., any coalition of q bits has influence at most $\Theta(q \cdot \log^2 n/n)$.

A.3 Definition of Adaptive OWF Families

Definition 9 ([58]). *Let $\mathbf{F} = \{\{f_{\mathsf{tag}} : X_{\mathsf{tag}} \longrightarrow Y_{\mathsf{tag}}\}_{\mathsf{tag} \in \{0,1\}^{\lambda}}\}_{\lambda \in \mathbb{N}}$ be a family of injective functions. We say that $\mathbf{F}$ is* an adaptive one-way function (AOWF) *family if:*

(1) *For every λ and $\mathsf{tag} \in \{0,1\}^{\lambda}$, the elements in X_{tag} can be randomly sampled efficiently, and there is a family of poly-size deterministic circuits $\mathcal{M} = \{M_{\lambda}\}_{\lambda \in \mathbb{N}}$ s.t. for every λ, $\mathsf{tag} \in \{0,1\}^{\lambda}$ and $x \in X_{\mathsf{tag}}$, $M_{\lambda}(\mathsf{tag}, x) = f_{\mathsf{tag}}(x)$.*

(2) Let $\mathcal{O}(\mathsf{tag},\cdot,\cdot)$ denote the oracle that, on input tag' and y outputs $f^{-1}_{\mathsf{tag}'}(y)$ if $\mathsf{tag}' \neq \mathsf{tag}, |\mathsf{tag}'| = |\mathsf{tag}|$, and $\perp$ otherwise. Then, for every PPT adversary $\mathcal{A}$ and every $\mathsf{tag} \in \{0,1\}^\lambda$, it holds that

$$\Pr\left[x \xleftarrow{\$} X_{\mathsf{tag}} : \mathcal{A}^{\mathcal{O}(\mathsf{tag},\cdot,\cdot)}\big(\mathsf{tag}, f_{\mathsf{tag}}(x)\big) = x\right] = \mathsf{negl}(\lambda) \ .$$

A.4 Instantiation of a Publicly Samplable Adaptive OWF Family

Pandey *et al.* presented two candidates for adaptive OWFs (AOWFs) in [58], based on discrete logarithm and factoring, respectively. The authors also claimed that AOWFs can be instantiated from RSA and Rabin functions.

In the following, we first give an instantiation of a collection of groups where the strong adaptive DDH assumption is conjectured to be true. For any security parameter $\lambda \in \mathbb{N}$, for any $\mathsf{tag} \in \{0,1\}^\lambda$, define the collection of groups $\{(q, p_{\mathsf{tag}}, g_{\mathsf{tag}}, G_{\mathsf{tag}})\}_{\mathsf{tag}\in\{0,1\}^\lambda}$ where p_{tag} is a 2λ bit prime whose first λ bits equal to tag, and q is a large (more than λ bits) prime factor of $p_{\mathsf{tag}} - 1$. $\langle g_{\mathsf{tag}} \rangle = G_{\mathsf{tag}}$ is a cyclic multiplicative group of prime order q. We assume that the standard DDH assumption holds for each group G_{tag}. Moreover, we set the corresponding inversion oracle $\mathcal{O}(\mathsf{tag},\cdot,\cdot)$ to be the discrete logarithm oracle $\mathcal{DL}(\mathsf{tag},\cdot,\cdot)$ that, on input tag' and y outputs x such that $g^x_{\mathsf{tag}'} = y$ if $\mathsf{tag} \neq \mathsf{tag}', |\mathsf{tag}'| = |\mathsf{tag}|$ and $\perp$ otherwise. We assume that accessing the discrete logarithm oracle of one or more group(s) gives no advantage on breaking the DDH assumption on the other group(s) (such assumption always holds in the generic group model). Formally, we have the following definition.

Definition 10. *We say that the DDH problem is adaptively hard w.r.t. the collection of groups $\{(q, p_{\mathsf{tag}}, g_{\mathsf{tag}}, G_{\mathsf{tag}})\}_{\mathsf{tag}\in\{0,1\}^\lambda}$, if for any PPT adversary $\mathcal{A}$ and every $\mathsf{tag} \in \{0,1\}^\lambda$, it holds that*

$$\Big| \Pr\left[x, y \xleftarrow{\$} \mathbb{Z}_q : \mathcal{A}^{\mathcal{DL}(\mathsf{tag},\cdot,\cdot)}\big(g_{\mathsf{tag}}, g^x_{\mathsf{tag}}, g^y_{\mathsf{tag}}, g^{xy}_{\mathsf{tag}}\big) = 1\right] - \\ - \Pr\left[x, y, z \xleftarrow{\$} \mathbb{Z}_q : \mathcal{A}^{\mathcal{DL}(\mathsf{tag},\cdot,\cdot)}\big(g_{\mathsf{tag}}, g^x_{\mathsf{tag}}, g^y_{\mathsf{tag}}, g^z_{\mathsf{tag}}\big) = 1\right] \Big| = \mathsf{negl}(\lambda) \ ,$$

where $\mathcal{DL}(\mathsf{tag},\cdot,\cdot)$ is the discrete logarithm oracle.

We note that the above strong adaptive DDH assumption implies the adaptive discrete logarithm assumption defined as:

Definition 11. *We say the adaptive discrete logarithm problem is hard w.r.t. the collection of groups $\{(q, p_{\mathsf{tag}}, g_{\mathsf{tag}}, G_{\mathsf{tag}})\}_{\mathsf{tag}\in\{0,1\}^\lambda}$ if for any PPT adversary $\mathcal{A}$ and every $\mathsf{tag} \in \{0,1\}^\lambda$ we have*

$$\Pr\left[y \xleftarrow{\$} \mathbb{Z}_q ; x \leftarrow \mathcal{A}^{\mathcal{DL}(\mathsf{tag},\cdot,\cdot)}\big(\mathsf{tag}, y\big) : g^x_{\mathsf{tag}} = y\right] = \mathsf{negl}(\lambda)$$

where $\mathcal{DL}(\mathsf{tag},\cdot,\cdot)$ is a discrete logarithm oracle that, on input tag' and y outputs x such that $g^x_{\mathsf{tag}'} = y$ if $\mathsf{tag} \neq \mathsf{tag}', |\mathsf{tag}'| = |\mathsf{tag}|$ and $\perp$ otherwise.

Let us now describe how to concretely instantiate a publicly samplable adaptive OWF family (PS-AOWF) based on the hardness of discrete logarithm w.r.t. the collection of groups $\{(q, p_{\mathsf{tag}}, g_{\mathsf{tag}}, G_{\mathsf{tag}})\}_{\mathsf{tag}\in\{0,1\}^\lambda}$. We first construct a simple image-mapping algorithm $\mathsf{IM}(\mathsf{tag}, \omega)$ with perfect samplability as follows: Let p_{tag} be a prime such that $p_{\mathsf{tag}} = kq + 1$ for some $k \in \mathbb{N}$. Given $\omega \in [0, q]$, $\mathsf{IM}(\mathsf{tag}, \omega)$ outputs $y = \omega^k \pmod{p_{\mathsf{tag}}}$. It is easy to see that y is always a valid image; therefore, the image-mapping algorithm has perfect samplability, i.e.

$$\Pr\left[\omega \leftarrow [0, q] : \mathsf{IM}(\mathsf{tag}, \omega) \in \mathbb{Z}_{p_{\mathsf{tag}}}\right] = 1 \ .$$

From Definition 11 it is easy to see that even an adversary is given the random coins ω can still only invert DL with negligible probability.

A.5 Standard Ideal Functionalities

For our VMPC definition we utilize standard versions of the ideal functionalities of a Bulletin Board and of a Secure Channel. For completeness we recall them below.

Formalizing the BB notion. We formally express the notion of a global bulletin board BB with consistent write/read operations in Figure 6.

Functionality $\mathcal{G}_{\mathsf{BB}}$

– Upon initialization, it creates an empty list L_{write}.

– Upon receiving $(\textsc{write}, \mathsf{sid}, \mathsf{pid}, x)$ from a server $S \in \mathcal{S}$, it appends the record x to L_{write} and sends $(\textsc{write}, \mathsf{sid}, x)$ to Sim.

– Upon receiving $(\textsc{Read}, \mathsf{sid})$ from any party P, it returns the data $(\mathsf{sid}, L_{\mathsf{write}})$ to the requester.

Fig. 6: The global setup bulletin board functionality $\mathcal{G}_{\mathsf{BB}}$.

Formalizing the $\mathcal{F}_{\mathsf{sc}}$ ideal functionality. In Figure 7 we recall the secure channels functionality $\mathcal{F}_{\mathsf{sc}}$ between each user and her client.

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

Upon receiving $(\textsc{send}, \mathsf{sid}, x, \mathsf{ID}')$ from entity ID, it sends $(\textsc{send}, \mathsf{sid}, \mathsf{ID}, |x|, \mathsf{ID}')$ to Sim and provides $(\textsc{sent}, \mathsf{sid}, \mathsf{ID}, x)$ to ID' as private delayed output.

Fig. 7: The ideal secure channel functionality $\mathcal{F}_{\mathsf{sc}}$.

B Sensitivity of CVZK soundness to the number of verifiers

In our definition of CVZK soundness, each verifier decides based on the transcript *and* its private state. We allow the adversary to corrupt up to t_2 verifiers and change their coins. Thus, if the adversary provides a transcript that is inconsistent with these verifiers' states, which depend on the original coins, it is enough if a single honest verifier detects the inconsistency. The intuition is that the probability *none* of the verifiers detects the inconsistency becomes very small, e.g., it drops exponentially with t_2. We could extend the definition of soundness by adding a parameter s that captures tolerance of failed verifications. Then, soundness would hold against adversaries that leave t_2-out-of-n verifiers honest, iff a subset of at least s-out-of-t_2 honest verifiers verify successfully. However, given the independency of verifications in our setting (every verifier decides based on her own private state), this extension would not be much more expressive security-wise in practice. For instance, in the case that each verifier contributes a single bit, requiring s-out-of-t_2 successful verifications carries the same error, e.g. $\epsilon_2 = 2^{-s}$, as asking exactly s successful verifications. Our constructions are adaptable in a straightforward manner to such an extended s-out-of-t_2 definition. For this reason, we will just focus on the above definitional setting where even a single honest failed verification invalidates a soundness attack.

C Supplementary Material for Section 4

C.1 Proof of Theorem 1

Proof. Let $\mathcal{I}_{\mathsf{corr}} \subset [n]$ s.t. $|\mathcal{I}_{\mathsf{corr}}| \leq t = \frac{n^{1-\frac{1}{\gamma}}}{\log^3 n}$ and an adversary $\mathcal{A}$. For the function $F : \{0,1\}^n \longrightarrow \left(\{0,1\}^{\frac{\lambda}{\log^2 \lambda}}\right)^{\log \lambda}$, the experiment $\mathbf{Expt}^{\mathsf{Coal}}_{(t,\mathcal{A},\mathcal{I}_{\mathsf{corr}})}(1^\lambda)$ is executed as follows:

1. Set $C := (c_1, \ldots, c_n) \leftarrow \mathbb{U}_n$;
2. $\mathcal{A}\big(\langle c_\ell \rangle_{\ell \in \mathcal{I}_{\mathsf{corr}}}\big)$ outputs $C' = (c'_1, \ldots, c'_n)$
   s.t. $\forall \ell \in [n] \setminus \mathcal{I}_{\mathsf{corr}} : c'_\ell = c_\ell$;
3. Return $(d_1, \ldots, d_m) \leftarrow F(C')$;

Namely, $C := (c_1, \ldots, c_n)$ is a uniformly random n-bit string, the bits of which are grouped as $(G_1, \ldots, G_{\log \lambda})$ and the adversary $\mathcal{A}$ statically tampers up to t bits of C. Let t_i, $i \in [\log \lambda]$ be the number of corrupted bits in group G_i, where $t_1 + \cdots + t_{\log \lambda} = t$.

Let $B_{i,1}, \ldots, B_{i,\lambda}$ denote the blocks inside the group G_i, i.e. $B_{i,j} = B_{(i-1)\lambda + j}$, $j \in [\lambda]$. Assume the adversary selects $t_{i,j}$ bits to corrupt for the block $B_{i,j}$, and $\forall i : \sum_{j=1}^{\lambda} t_{i,j} = t_i$. Let $\mathsf{Inf}_{i,j}$ be the event that over the input coins of $B_{i,j}$ the adversary $\mathcal{A}$ influences the output bit $b_{i,j} \leftarrow f_{\mathsf{res}}(B_{i,j})$, where $i \in [\log \lambda], j \in [\lambda]$. By the $(\Theta(\frac{\log^2 m}{m}))$-strong resilience of f_{res}, if we set $m = \frac{n}{\lambda \log \lambda}$ (the input block length), then each corrupted bit in $B_{i,j}$ has probability $\Theta(\frac{\log^3 n}{n^{1-\frac{1}{\gamma}}})$ to influence the output bit of $B_{i,j}$. By the union bound, the latter implies that

$$\Pr\left[\mathsf{Inf}_{i,j}\right] = \Theta\Big(t_{i,j} \cdot \frac{\log^3 n}{n^{1-\frac{1}{\gamma}}}\Big) \,. \tag{3}$$

Next, we define as G_i the event that the adversary does not influence any of the λ bits output of group G_i. We will prove that the events $\mathsf{G}_1, \ldots, \mathsf{G}_{\log \lambda}$ meet the conditions in Definition 4 that set F as a $\left(\frac{\lambda}{\log^2\lambda}, \log\lambda, \frac{n^{1-\frac{1}{\gamma}}}{\log^3 n}\right)$-coalescence function w.r.t. $\mathbb{U}_n$.

Condition (i). $\Pr[\wedge_{i=1}^{\log\lambda}\neg\mathsf{G}_i] = \mathsf{negl}(\lambda)$: by Eq. (3), the union bound and the fact that $\sum_{j=1}^{\lambda} t_{i,j} \leq t_i$, we have that the probability that $\mathcal{A}$ influences at least one of the λ bits output of group G_i

$$\Pr[\neg\mathsf{G}_i] = \Pr[\vee_{j=1}^{\lambda}\mathsf{Inf}_{i,j}] \leq \sum_{j=1}^{\lambda}\Pr[\mathsf{Inf}_{i,j}] =$$

$$= \sum_{j=1}^{\lambda}\Theta\Big(t_{i,j}\cdot\frac{\log^3 n}{n^{1-\frac{1}{\gamma}}}\Big) = \Theta\Big(t_i\cdot\frac{\log^3 n}{n^{1-\frac{1}{\gamma}}}\Big), \text{ for any } i\in[\log\lambda]\ .$$

Recall that $\mathcal{A}$ can statically corrupt up to t bits, which implies that the events $\mathsf{G}_1,\ldots\mathsf{G}_{\log\lambda}$ are mutually independent and their probabilities are affected only by the arrangement $t_1,\ldots,t_{\log\lambda}$ of corrupted bit positions that $\mathcal{A}$ selects. Therefore, the probability that for every $i\in[\log\lambda]$, at least one of the output bits of G_i is affected by some corrupted bits is bounded by

$$\Pr[\wedge_{i=1}^{\log\lambda}\neg\mathsf{G}_i] = \prod_{i=1}^{\log\lambda}\Pr[\neg\mathsf{G}_i] \leq \prod_{i=1}^{\log\lambda}\Theta\Big(t_i\cdot\frac{\log^3 n}{n^{1-\frac{1}{\gamma}}}\Big)\ .$$

Now, we make use of the fact that the product $\prod_{i=1}^{m} t_i$ under the restriction $\sum_{i=1}^{m} t_i = t$ is maximized at the value $\left(\frac{t}{m}\right)^m$ to we get that

$$\Pr[\wedge_{i=1}^{\log\lambda}\neg\mathsf{G}_i] \leq \prod_{i=1}^{\log\lambda}\Big(\Theta\Big(t_i\cdot\frac{\log^3 n}{n^{1-\frac{1}{\gamma}}}\Big)\Big) = \prod_{i=1}^{\log\lambda} t_i\prod_{i=1}^{\log\lambda}\Big(\Theta\Big(\frac{\log^3 n}{n^{1-\frac{1}{\gamma}}}\Big)\Big) \leq$$

$$\leq \Big(\frac{t}{\log\lambda}\Big)^{\log\lambda}\prod_{i=1}^{\log\lambda}\Big(\Theta\Big(\frac{\log^3 n}{n^{1-\frac{1}{\gamma}}}\Big)\Big)\ .$$

Hence, by setting $t = \frac{n^{1-\frac{1}{\gamma}}}{\log^3 n}$, and since $\log\lambda = \Theta(\log n)$, we have that

$$\begin{aligned}\Pr[\wedge_{i=1}^{\log\lambda}\neg\mathsf{G}_i] &\leq \Big(\Theta\Big(\frac{n^{1-\frac{1}{\gamma}}}{\log^4 n}\Big)\Big)^{\log\lambda}\cdot\prod_{i=1}^{\log\lambda}\Big(\Theta\Big(\frac{\log^3 n}{n^{1-\frac{1}{\gamma}}}\Big)\Big) = \\ &= \prod_{i=1}^{\log\lambda}\Theta\Big(\frac{1}{\log n}\Big) = O(1/\log^{\log\lambda} n) = O(2^{-\log\log n\log\lambda}) = \mathsf{negl}(\lambda)\end{aligned} \tag{4}$$

Condition (ii). For all $i\in[\log\lambda]$, the random variable $(d_i|\mathsf{G}_i)$ is statistically $\mathsf{negl}(\lambda)$*-close to* $\mathbb{U}_{\frac{\lambda}{\log^2\lambda}}$*:* we define the event Fail_i that the rejection sampling process fails for a bit of group G_i. We will show that for every G_i, on condition that G_i occurs, then Fail_i happens with negligible probability. Specifically, assume that G_i occurs, so the group G_i is unaffected. Now, since the non-corrupted bits of G_i are random and the bias property of f_{res}, we have that the output bits $b_{i,j} \leftarrow f_{\mathsf{res}}(B_{i,j})$, $i \in [\lambda]$ have a bias $\frac{1}{10}$. Therefore, by providing the rejection sampling process with input $(b_{i,1}, \ldots, b_{i,\lambda})$, the probability that a pair $(b_{i,j}, b_{i,j+1})$ consists of two non-equal bits is

$$\Big(\frac{1}{2}+\frac{1}{10}\Big)\cdot\Big(\frac{1}{2}-\frac{1}{10}\Big)+\Big(\frac{1}{2}-\frac{1}{10}\Big)\cdot\Big(\frac{1}{2}+\frac{1}{10}\Big)=\frac{12}{25}\ .$$

Thus, the probability of $\log^2\lambda$ consecutive fails is $\Big(1-\frac{12}{25}\Big)^{-\log^2\lambda}$. If up to $\log^2\lambda$ iterations are needed, then at most $2\log^2\lambda$ input bits are "consumed" in order to output an unbiased bit. As a result, the probability that for the group G_i, the process will succeed in outputting a $\frac{\lambda}{\log^2\lambda}$-bit string d_i is at least

$$\Big(1-\Big(\frac{13}{25}\Big)^{-\log^2\lambda}\Big)^{\frac{\lambda}{2\log^2\lambda}} \geq 1-\frac{\lambda}{2\log^2\lambda}\Big(\frac{13}{25}\Big)^{-\log^2\lambda}=1-\mathsf{negl}(\lambda)\ .$$

We conclude that

$$\forall i\in[\log\lambda] : \Pr[\mathsf{Fail}_i|\mathsf{G}_i]=\mathsf{negl}(\lambda)\ . \tag{5}$$

Assume now that $\mathsf{G}_i \wedge (\neg\mathsf{Fail}_i)$ occurs. Then, the output bits $(b_{i,1}, \ldots, b_{i,\lambda})$ for G_i are not influenced, so by the properties of f_{res} they are random and have a bias $\frac{1}{10}$. When $(b_{i,1}, \ldots, b_{i,\lambda})$ is provided as input to the rejection sampling process, we have that

$$\Pr[b_{i,j}=1, b_{i,j+1}=0]=\Pr[b_{i,j}=0, b_{i,j+1}=1],$$

so given that $b_{i,j}\neq b_{i,j+1}$ the corresponding produced bit of d_i is unbiased. Thus, when the process succeeds, the output d_i consists of $\frac{\lambda}{\log^2\lambda}$ random and unbiased bits. As a result, for all for all $i\in\{1,\ldots,\log\lambda\}$, $(d_i|\mathsf{G}_i\wedge(\neg\mathsf{Fail}_i))\sim\mathbb{U}_{\frac{\lambda}{\log^2\lambda}}$.

By the above and applying Eq. (5), we compute the statistical distance between $d_i|\mathsf{G}_i$ and $\mathbb{U}_{\frac{\lambda}{\log^2\lambda}}$ as follows:

$$\begin{aligned}
&\Delta\big[d_i|\mathsf{G}_i,\mathbb{U}_{\frac{\lambda}{\log^2\lambda}}\big]=\\
&=\sum_{z\in\{0,1\}^{\frac{\lambda}{\log^2\lambda}}}\Big|\ \Pr[d_i|\mathsf{G}_i=z]-2^{-\frac{\lambda}{\log^2\lambda}}\ \Big|=\\
&=\sum_{z\in\{0,1\}^{\frac{\lambda}{\log^2\lambda}}}\Big|\ \Pr[\neg\mathsf{Fail}_i|\mathsf{G}_i]\cdot\Pr[d_i|\mathsf{G}_i=z_i|\mathsf{G}_i\wedge(\neg\mathsf{Fail}_i)]-2^{-\frac{\lambda}{\log^2\lambda}}\ \Big|\\
&=\sum_{z\in\{0,1\}^{\frac{\lambda}{\log^2\lambda}}}\Big|\ (1-\Pr[\mathsf{Fail}_i|\mathsf{G}_i])\cdot2^{-\frac{\lambda}{\log^2\lambda}}+2^{-\frac{\lambda}{\log^2\lambda}}\ \Big|\leq\\
&\leq\sum_{z\in\{0,1\}^{\frac{\lambda}{\log^2\lambda}}}\Big|\ (1-\mathsf{negl}(\lambda))\cdot2^{-\frac{\lambda}{\log^2\lambda}}-2^{-\frac{\lambda}{\log^2\lambda}}\ \Big|=\mathsf{negl}(\lambda)\ .
\end{aligned}$$

As a result, condition (ii) also holds which implies that F is a $\left(\frac{\lambda}{\log^2\lambda}, \log\lambda, \frac{n^{1-\frac{1}{\gamma}}}{\log^3 n}\right)$-coalescence function w.r.t. $\mathbb{U}_n$.

It remains to show that F satisfies completeness and efficient samplability.

Completeness: Recall that the output of $F(C)$ is determined by $\lambda\log\lambda$ applications of the resilient function f_{res} on each of the blocks $B_{i,j}$ that C was split. If C is sampled from $\mathbb{U}_n$, then each $B_{i,j}$ is also uniform. By the $(\Theta(\log^2 n/n))$-strong resilience of f_{res} (see Definition 1), we have that the probability that $f_{\mathsf{res}}(B_{i,j})$ is undetermined after fixing all inputs bits (i.e. the bits outside $S = \emptyset$) uniformly at random, is

$$I_{\emptyset}(f_{\mathsf{res}}) = I_0(f_{\mathsf{res}}) \leq (\Theta(\log^2 n/n)) \cdot 0 = 0\ .$$

As a result, every output bit $b_{i,j}$ will be random with bias $\frac{1}{10}$. As shown previously, for every i, the rejection sampling process will fail to output a random unbiased bit sequence d_i only with $\mathsf{negl}(\lambda)$ probability. Therefore, the statistical distance between $F(\mathbb{U}_n) = (d_1, \ldots, d_{\log\lambda})$ and $\big(\mathbb{U}_{\frac{\lambda}{\log^2\lambda}}\big)^{\log\lambda}$ is bounded by

$$\begin{aligned}
&\Delta\big[(d_1,\ldots,d_{\log\lambda}), \big(\mathbb{U}_{\frac{\lambda}{\log^2\lambda}}\big)^{\log\lambda}\big] = \\
&= \sum_{(z_1,\ldots,z_{\log\lambda})} \Big| \Pr[(d_1,\ldots,d_{\log\lambda}) = (z_1,\ldots,z_{\log\lambda})] - 2^{-(\frac{\lambda}{\log^2\lambda})^{\log\lambda}} \Big| = \\
&= \sum_{(z_1,\ldots,z_{\log\lambda})} \Big| \prod_{i=1}^{\log\lambda} \Pr[d_i = z_i] - 2^{-\frac{\lambda}{\log\lambda}} \Big| = \\
&= \sum_{(z_1,\ldots,z_{\log\lambda})} \Big| \prod_{i=1}^{\log\lambda} (\Pr[\neg\mathsf{Fail}_i] \cdot \Pr[d_i = z_i | \neg\mathsf{Fail}_i]) - 2^{-\frac{\lambda}{\log\lambda}} \Big| = \\
&= \sum_{(z_1,\ldots,z_{\log\lambda})} \Big| \prod_{i=1}^{\log\lambda} (\Pr[\neg\mathsf{Fail}_i] \cdot \Pr[d_i = z_i | \mathsf{G}_i]) - 2^{-\frac{\lambda}{\log\lambda}} \Big| = \\
&= \sum_{(z_1,\ldots,z_{\log\lambda})} \Big| \prod_{i=1}^{\log\lambda} (1 - \Pr[\mathsf{Fail}_i]) \cdot 2^{-(\frac{\lambda}{\log^2\lambda})^{\log\lambda}} + 2^{-\frac{\lambda}{\log\lambda}} \Big| \leq \\
&\leq \sum_{(z_1,\ldots,z_{\log\lambda})} \Big| (1 - \mathsf{negl}(\lambda)) \cdot 2^{-\frac{\lambda}{\log\lambda}} - 2^{-\frac{\lambda}{\log\lambda}} \Big| = \mathsf{negl}(\lambda)\ .
\end{aligned}$$

Efficient samplability: before presenting our $\mathsf{Sample}(\cdot)$ algorithm for F, let us first describe for some given $d_i = (d_{i,1}, \ldots, d_{i,\frac{\lambda}{\log^2\lambda}})$, the format of a bit sequence $\mathbf{b}_i = (b_{i,1}, \ldots, b_{i,\lambda})$ that if provided as input to the rejection sampling process, then the output is d_i.

First, in order for every bit $d_{i,k}$ to be produced, the pair $(d_{i,k}, 1-d_{i,k})$, where $k \in [\frac{\lambda}{\log^2\lambda}]$ must appear on $\mathbf{b}_i$.

Second, strings $y_{i,1}, \ldots, y_{i,\frac{\lambda}{\log^2\lambda}}$ of pairs of equal bits (i.e., 00 or 11) preceding the pairs $(d_1, 1-d_1), \ldots, (d_{\frac{\lambda}{\log^2\lambda}} 1 - d_{\frac{\lambda}{\log^2\lambda}})$ may appear on $\mathbf{b}_i$ (the rejection sampling performs a repetition when running into such pairs). Let $2\ell_{i,k} \geq 0$ be the length of each $y_{i,k}$, i.e. $y_{i,k}$ consists of $\ell_{i,k}$ pairs of equal bits. Finally, $\mathbf{b}_i$ may have a suffix z_i of arbitrary bits and length $\ell_{i,z} = \lambda - 2\frac{\lambda}{\log^2\lambda} + \sum_{k=1}^{\frac{\lambda}{\log^2\lambda}} \ell_{i,k}$.

We define the set $S[d_i]$ of all λ-bit sequences that when input to the rejection sampling process, they produce d_i. By the above, $S[d_i]$ can be formally described as

$$\begin{aligned} S[d_i] := \quad & \bigcup_{\ell_{i,1},\ldots,\ell_{i,\lambda/\log^2\lambda},\ell_{i,z}\colon \ell_{i,z}+\sum_{k=1}^{\frac{\lambda}{\log^2\lambda}} 2\ell_{i,k}=\lambda-2\frac{\lambda}{\log^2\lambda}} \Big((\{00,11\})^{\ell_1} || (d_{i,1}, 1-d_{i,1}) || \cdots \\ & \cdots || (\{00,11\})^{\ell_{\lambda/\log^2\lambda}} || (d_{i,\frac{\lambda}{\log^2\lambda}}, 1 - d_{i,\frac{\lambda}{\log^2\lambda}}) || (\{0,1\})^{\ell_z} \Big) \ . \end{aligned}$$

Next, we define the $\mathsf{Sample}(\cdot)$ algorithm for F. Intuitively on input $(d_1, \ldots, d_{\log\lambda})$, Sample chooses a random $\mathbf{b}_i = (b_{i,1}, \ldots, b_{i,\lambda}) \in S[d_i]$ for every d_i and then performs uniformly random rejection sampling from $\big(\mathbb{U}_{\frac{n}{\lambda\log\lambda}}\big)^{\lambda}$ until it hits a block sequence $(B_{(i-1)\lambda+1}, \ldots, B_{i\lambda})$ s.t. $(b_{i,1}, \ldots, b_{i,\lambda}) \leftarrow \big(f_{\mathsf{res}}(B_{(i-1)\lambda+1}), \ldots, f_{\mathsf{res}}(B_{i\lambda})\big)$ (or abort if rejection sampling fails). In case of success, it outputs all $\log\lambda$ suitable block sequences.

The algorithm Sample on input $d_1, \ldots, d_{\log\lambda}$ executes the following steps:

1. **For** $i = 1, \ldots, \log\lambda$,
   (a) Choose random string lengths $\ell_{i,1}, \ldots, \ell_{i,\frac{\lambda}{\log^2\lambda}}, \ell_{i,z}$ such that $\ell_{i,z} + \sum_{k=1}^{\frac{\lambda}{\log^2\lambda}} 2\ell_{i,k} = \lambda - 2\frac{\lambda}{\log^2\lambda}$.
   (b) For $k = 1, \ldots, \frac{\lambda}{\log^2\lambda}$ choose $\ell_{i,k}$ random pairs of equal pairs and concatenate them to create substring $y_{i,k}$.
   (c) Choose a random string z_i of length $\ell_{i,z}$.
   (d) Set the λ-bit string $\mathbf{b_i} \in S[d_i]$ as

$$\begin{aligned} \mathbf{b}_i = (b_{i,1}, \ldots, b_{i,\lambda}) := y_{i,1} || (d_{i,1}, 1 - d_{i,1}) || \cdots \\ \cdots || y_{i,\frac{\lambda}{\log^2\lambda}} || (d_{i,\frac{\lambda}{\log^2\lambda}}, 1 - d_{i,\frac{\lambda}{\log^2\lambda}}) || z_i \ . \end{aligned}$$

   (e) **For** $j = 1, \ldots, \lambda$, pick a uniformly random $\frac{n}{\lambda\log\lambda}$-bit string $u_{i,j} \leftarrow U_{\frac{n}{\lambda\log\lambda}}$ and check whether $f_{\mathsf{res}}(u_{i,j}) = b_{i,j}$. If yes, then set $B_{(i-1)\lambda+j} \leftarrow u_j$; otherwise, repeat by sampling a fresh $(\frac{n}{\lambda}\log\lambda)$-bit string $u_{i,j}$ until the equality $f_{\mathsf{res}}(u_{i,j}) = b_{i,j}$ holds. If λ iterations fail, then abort.
   (f) Set group block sequence G_i as $G_i := (B_{(i-1)\lambda+1}, \ldots, B_{i\lambda})$.
2. Output $C = (c_1, \ldots, c_n) := (G_1, \ldots, G_{\log\lambda})$.

We now show the two properties of $\mathsf{Sample}(\cdot)$ that guarantee the efficient samplability of F.

(i). The following probability is $1 - \mathsf{negl}(\lambda)$:

$$\Pr_{(d_1,\ldots,d_{\log\lambda}) \leftarrow \big(\mathbb{U}_{\frac{\lambda}{\log^2\lambda}}\big)^{\log\lambda}} \big[C \leftarrow \mathsf{Sample}(d_1, \quad \ldots, d_{\log\lambda}) : F(C) = (d_1, \ldots, d_{\log\lambda}) \big] \ .$$

We observe that if $\mathsf{Sample}(d_1, \ldots, d_{\log \lambda})$ does not fail, i.e. for every $i \in [\log \lambda]$ and $j \in [\lambda]$, the algorithm manages to sample a suitable block $B_{(i-1)\lambda+j}$ (Step 1.(e)), then the output C is such that $F(C) = (d_1, \ldots, d_{\log \lambda})$. This is because for every $i \in [\log \lambda]$. steps 1.(a)-(d) guarantee that $\mathbf{b}_i \in S[d_i]$. What remains to show is that the success probability of $\mathsf{Sample}(d_1, \ldots, d_{\log \lambda})$ is $1 - \mathsf{negl}(\lambda)$.

By the constant bias $\frac{1}{10}$ of f_{res} we have that for every $i \in [\log \lambda$ and $j \in [\lambda]$

$$\Pr\left[u_{i,j} \leftarrow U_{\frac{n}{\lambda \log \lambda}} : f_{\mathsf{res}}(u_{i,j}) \neq b_{i,j}\right] \leq \frac{1}{2} + \frac{1}{10} = \frac{3}{5} \ .$$

Therefore, the probability that Step 1.(e) will fail for $b_{i,j}$ after λ iterations is no more than $(\frac{3}{5})^{\lambda}$. Thus, by the union bound the probability that Step 1.(e) will not fail for any $i \in [\log \lambda]$ and $j \in [\lambda]$ is at least

$$1 - \lambda \log \lambda \left(\frac{3}{5}\right)^{\lambda} = 1 - \mathsf{negl}(\lambda) \ .$$

(ii). *The distribution* $\mathsf{Sample}\big((\mathbb{U}_{\frac{\lambda}{\log^2 \lambda}})^{\log \lambda}\big)$ *is statistically* $\mathsf{negl}(\lambda)$*-close to* $\mathbb{U}_n$: as a first step, we observe that for every two distinct strings $d_i := (d_{i,1}, \ldots, d_{i,\lambda})$, $d'_i := (d'_{i,1}, \ldots, d'_{i,\lambda})$ the elements (λ-bit strings) in the sets $S[d_i]$ and $S[d'_i]$ are in 1-1 correspondence. This holds because for every λ-bit string

$$\begin{aligned} \mathbf{b}_i = (b_{i,1}, \ldots, b_{i,\lambda}) &:= y_{i,1}||(d_{i,1}, 1 - d_{i,1})|| \cdots \\ &\cdots ||y_{i,\frac{\lambda}{\log^2 \lambda}}||(d_{i,\frac{\lambda}{\log^2 \lambda}}, 1 - d_{i,\frac{\lambda}{\log^2 \lambda}})||z_i \ . \end{aligned}$$

that is in $S[d_i]$, then the λ-bit string

$$\begin{aligned} \mathbf{b}'_i = (b'_{i,1}, \ldots, b'_{i,\lambda}) &:= y_{i,1}||(d'_{i,1}, 1 - d'_{i,1})|| \cdots \\ &\cdots ||y_{i,\frac{\lambda}{\log^2 \lambda}}||(d'_{i,\frac{\lambda}{\log^2 \lambda}}, 1 - d'_{i,\frac{\lambda}{\log^2 \lambda}})||z_i \ . \end{aligned}$$

is in $S[d'_i]$. It directly follows that for any two d_i, d'_i the sets $S[d_i]$ and $S[d'_i]$ are of equal size.

Next, we observe that every two corresponding $\mathbf{b}_i$ and $\mathbf{b}'_i$ differ only at a subset of the pairs $(d_{i,k}, 1 - d_{i,k})$, $(d'_{i,k}, 1 - d'_{i,k})$, $k \in [\frac{\lambda}{\log^2 \lambda}]$. Namely, it could be the case that $(d_{i,k}, 1 - d_{i,k}) = (0, 1)$ and $(d'_{i,k}, 1 - d'_{i,k}) = (1, 0)$ or vice versa. In any case, both $\mathbf{b}_i$ and $\mathbf{b}'_i$ have the same number of 0s and 1s.

Let $H[\mathbf{b}_i]$ denote the set of suitable block sequences $(B_{(i-1)\lambda+1}, \ldots, B_{i\lambda})$ for $\mathbf{b}_i$ i.e. $(b_{i,1}, \ldots, b_{i,\lambda}) \leftarrow \big(f_{\mathsf{res}}(B_{(i-1)\lambda+1}), \ldots, f_{\mathsf{res}}(B_{i\lambda})\big)$. We will show that since $\mathbf{b}_i$ and $\mathbf{b}'_i$ have the same number of 0s and 1s, $H[\mathbf{b}_i]$ and $H[\mathbf{b}'{}_i]$ are of equal size.

In particular, it holds that

$$\begin{aligned}
|H[\mathbf{b}_i]| &= \prod_{j=1}^{\lambda} \big|\,\{\, B_{(i-1)\lambda+j} : f_{\mathsf{res}}(B_{(i-1)\lambda+j}) = b_{i,j} \,\}\,\big| = \\
&= \prod_{j:b_{i,j}=0} \big|\,\{\, B_{(i-1)\lambda+j} : f_{\mathsf{res}}(B_{(i-1)\lambda+j}) = 0 \,\}\,\big| \cdot \\
&\quad \cdot \prod_{j:b_{i,j}=1} \big|\,\{\, B_{(i-1)\lambda+j} : f_{\mathsf{res}}(B_{(i-1)\lambda+j)} = 1 \,\}\,\big| = \\
&= \prod_{j:b'_{i,j}=0} \big|\,\{\, B'_{(i-1)\lambda+j} : f_{\mathsf{res}}(B'_{(i-1)\lambda+j}) = 0 \,\}\,\big| \cdot \\
&\quad \cdot \prod_{j:b'_{i,j}=1} \big|\,\{\, B'_{(i-1)\lambda+j} : f_{\mathsf{res}}(B'_{(i-1)\lambda+j}) = 1 \,\}\,\big| = \\
&= \prod_{j=1}^{\lambda} \big|\,\{\, B'_{(i-1)\lambda+j} : f_{\mathsf{res}}(B'_{(i-1)\lambda+j}) = b'_{i,j} \,\}\,\big| = |H[\mathbf{b}'_i]| \,.
\end{aligned} \tag{6}$$

As shown previously, the probability of failure (denoted by the event Fail) of $\mathsf{Sample}(d_1, \ldots, d_{\log\lambda})$ is $\mathsf{negl}(\lambda)$. We will prove that when $\mathsf{Sample}\big((\mathbb{U}_{\frac{\lambda}{\log^2\lambda}})$ does not fail, then the output $C = (c_1, \ldots, c_n) := (G_1, \ldots, G_{\log\lambda})$ follows $\mathbb{U}_n$. By performing similar computation steps as previously in the proof (e.g. see completeness), this suffices for proving that $\mathsf{Sample}\big((\mathbb{U}_{\frac{\lambda}{\log^2\lambda}})^{\log\lambda}\big)$ is statistically $\mathsf{negl}(\lambda)$-close to $\mathbb{U}_n$.

For every $G_i := (B_{(i-1)\lambda+1}, \ldots, B_{i\lambda})$, let $E[G_i]$ be the event that $\mathsf{Sample}(\ldots, d_i, \ldots)$, where d_i is random, fixes at Step 1.(d) the unique string $\mathbf{b}_i$ s.t. $G_i \in H[\mathbf{b}_i]$. Recall that by construction, $\mathsf{Sample}(d_1, \ldots, d_{\log\lambda})$ chooses the respective strings $\mathbf{b}_1, \ldots, \mathbf{b}_{\log\lambda}$ uniformly at random from the sets $S[d_1], \ldots, S[d_{\log\lambda}]$, respectively (see Steps 1(a)-(d)). Hence, by Eq. (6), we have that for any two distinct $C := (G_1, \ldots, G_{\log\lambda})$, $C' := (G'_1, \ldots, G'_{\log\lambda})$

$$
\begin{aligned}
&\Pr\left[(d_1,\ldots,d_{\log\lambda}) \leftarrow \left(\mathbb{U}_{\frac{\lambda}{\log^2\lambda}}\right)^{\log\lambda} : C \leftarrow \mathsf{Sample}(d_1,\ldots,d_{\log\lambda})\right] = \\
&= \prod_{i=1}^{\log\lambda} \Pr\left[d_i \leftarrow \mathbb{U}_{\frac{\lambda}{\log^2\lambda}} : (\ldots,G_i,\ldots) \leftarrow \mathsf{Sample}(\ldots,d_i,\ldots)\right] = \\
&= \prod_{i=1}^{\log\lambda} \Pr\left[d_i \leftarrow \mathbb{U}_{\frac{\lambda}{\log^2\lambda}} : E[G_i] \wedge G_i \in H[\mathbf{b}_i] \wedge \mathbf{b}_i \in S[d_i]\right] = \\
&= \prod_{i=1}^{\log\lambda} \Pr\left[d_i \leftarrow \mathbb{U}_{\frac{\lambda}{\log^2\lambda}} : G_i \in H[\mathbf{b}_i] \wedge \mathbf{b}_i \in S[d_i]\right]\cdot \\
&\quad \cdot \Pr\left[d_i \leftarrow \mathbb{U}_{\frac{\lambda}{\log^2\lambda}} : E[G_i] \;\middle|\; G_i \in H[\mathbf{b}_i] \wedge \mathbf{b}_i \in S[d_i]\right] = \\
&= \prod_{i=1}^{\log\lambda} 2^{-\frac{\lambda}{\log^2\lambda}} \cdot \frac{1}{|S[d_i]|} \cdot \frac{1}{|H[\mathbf{b}_i]|} = \\
&= \prod_{i=1}^{\log\lambda} 2^{-\frac{\lambda}{\log^2\lambda}} \cdot \frac{1}{|S[d'_i]|} \cdot \frac{1}{|H[\mathbf{b}'_i]|} = \\
&= \prod_{i=1}^{\log\lambda} \Pr\left[d'_i \leftarrow \mathbb{U}_{\frac{\lambda}{\log^2\lambda}} : G'_i \in H[\mathbf{b}'_i] \wedge \mathbf{b}'{}_i \in S[d'_i]\right]\cdot \\
&\quad \cdot \Pr\left[d'_i \leftarrow \mathbb{U}_{\frac{\lambda}{\log^2\lambda}} : E[G'_i] \;\middle|\; G'_i \in H[\mathbf{b}'_i] \wedge \mathbf{b}'_i \in S[d'_i]\right] = \\
&= \prod_{i=1}^{\log\lambda} \Pr\left[d'_i \leftarrow \mathbb{U}_{\frac{\lambda}{\log^2\lambda}} : E[G'_i] \wedge G'_i \in H[\mathbf{b}'_i] \wedge \mathbf{b}'{}_i \in S[d'_i]\right] = \\
&= \prod_{i=1}^{\log\lambda} \Pr\left[d'_i \leftarrow \mathbb{U}_{\frac{\lambda}{\log^2\lambda}} : (\ldots,G'_i,\ldots) \leftarrow \mathsf{Sample}(\ldots,d'_i,\ldots)\right] = \\
&= \Pr\left[(d'_1,\ldots,d'_{\log\lambda}) \leftarrow \left(\mathbb{U}_{\frac{\lambda}{\log^2\lambda}}\right)^{\log\lambda} : C' \leftarrow \mathsf{Sample}(d'_1,\ldots,d'_{\log\lambda})\right] .
\end{aligned}
\tag{7}
$$

By Eq. (7), when $\mathsf{Sample}\big(\big(\mathbb{U}_{\frac{\lambda}{\log^2\lambda}}\big)^{\log\lambda}\big)$ does not fail, each $C \in \{0,1\}^n$ has the same probability of being output. Namely,

$$\mathsf{Sample}\big(\big(\mathbb{U}_{\frac{\lambda}{\log^2\lambda}}\big)^{\log\lambda}\big)\big|(\neg\mathsf{Fail}) \sim \mathbb{U}_n \; .$$

Since $\Pr[\mathsf{Fail}] = \mathsf{negl}(\lambda)$, we have that

$$\varDelta\big[\mathsf{Sample}\big(\big(\mathbb{U}_{\frac{\lambda}{\log^2\lambda}}\big)^{\log\lambda}\big), \mathbb{U}_n\big] = \mathsf{negl}(\lambda) \; .$$

□

C.2 A commitment scheme secure against helper-aided adversaries

We denote a commitment scheme CS as a triple of efficient algorithms $(\mathsf{CS.Gen}, \mathsf{CS.Com}, \mathsf{CS.Ver})$, where (i) the generator algorithm $\mathsf{CS.Gen}(1^\lambda)$ outputs a commitment key ck, (ii) the commitment algorithm $\mathsf{CS.Com}_{\mathsf{ck}}(M)$ outputs a commitment c to message M, and (iii) the verification algorithm $\mathsf{CS.Ver}_{\mathsf{ck}}(c, (M, r))$ outputs 1 accepting M as an opening of c given decommitment information (M, r), or 0, otherwise.

In this subsection, we present a commitment scheme that is perfectly binding and computationally hiding against helper-aided PPT adversaries, given the function family $\mathbf{F} = \{f_{\mathsf{tag}} : \mathbb{Z}_q \longrightarrow G_{\mathsf{tag}}\}_{\mathsf{tag}\in\{0,1\}^\lambda}$, where $f_{\mathsf{tag}}(x) := g_{\mathsf{tag}}^x$, such that the DDH is adaptively hard w.r.t. the collection of groups $\{(q, p_{\mathsf{tag}}, g_{\mathsf{tag}}, G_{\mathsf{tag}})\}_{\mathsf{tag}\in\{0,1\}^\lambda}$ as instantiated in A.4.

Description. We consider a subset of k tags, $\{\mathsf{tag}_1^*, \ldots, \mathsf{tag}_k^*\} \subset \{0,1\}^\lambda$. We note that in the VMPC setting, these tags will refer to the k MPC servers that will jointly setup the commitment key.

For tag_i^*, $i \in [k]$, let $G_{\mathsf{tag}_i^*}$ be the underlying group of order q with generator $g_{\mathsf{tag}_i^*}$. Our scheme utilizes the following tools:

1. For every tag_i^*, $i \in [k]$, the (lifted) ElGamal encryption scheme $(g_{\mathsf{tag}_i^*}^{r_i}, g_{\mathsf{tag}_i^*}^{M_i} h_{\mathsf{tag}_i^*}^{r_i})$, where $(q, p_{\mathsf{tag}_i^*}, g_{\mathsf{tag}_i^*}, h_{\mathsf{tag}_i^*})$ is the public key and message M_i is encrypted under randomness r_i.
2. A k-out-of-k additive secret sharing scheme, i.e., $M = M_1 + M_2 + \cdots + M_k \bmod q$.

Our commitment scheme $\mathsf{CS}^{\mathbf{F}} = (\mathsf{CS}^{\mathbf{F}}.\mathsf{Gen}, \mathsf{CS}^{\mathbf{F}}.\mathsf{Com}, \mathsf{CS}^{\mathbf{F}}.\mathsf{Ver})$ w.r.t. $\{\mathsf{tag}_1^*, \ldots, \mathsf{tag}_k^*\}$ is specified as follows:

- $\mathsf{CS.Gen}^{\mathbf{F}}(1^\lambda)$:
  1. For each $i \in [k]$, choose a random value $t_i \overset{\$}{\leftarrow} \mathbb{Z}_q$ and set the partial commitment key $\mathsf{ck}_i = (q, p_{\mathsf{tag}_i^*}, g_{\mathsf{tag}_i^*}, h_{\mathsf{tag}_i^*} := g_{\mathsf{tag}_i^*}^{t_i})$.
  2. Output $\mathsf{ck} = (\mathsf{ck}_1, \ldots, \mathsf{ck}_k)$.

- $\mathsf{CS.Com}_{\mathsf{ck}}^{\mathbf{F}}(M)$: commit to M as below
  1. Split M into k shares $M_1, \ldots, M_k$ s.t. $M = \sum_{i=1}^k M_i \bmod q$.
  2. For each $i \in [k]$, choose randomness $r_i \overset{\$}{\leftarrow} \mathbb{Z}_q$ and compute the ElGamal ciphertext $\psi_i := (g_{\mathsf{tag}_i^*}^{r_i}, g_{\mathsf{tag}_i^*}^{M_i} h_{\mathsf{tag}_i^*}^{r_i})$.
  3. Output the commitment $\mathbf{c} := (\psi_1, \ldots, \psi_k)$. The decommitment information is $\langle M_i, r_i \rangle_{i\in[k]}$.

- $\mathsf{CS.Ver}_{\mathsf{ck}}^{\mathbf{F}}(\mathbf{c}, (M, \langle M_i, r_i \rangle_{i\in[k]}))$: parse $\mathbf{c} := (\psi_1, \ldots, \psi_k)$ and output 1 iff the following checks hold
  1. $\sum_{i\in[k]} M_i \equiv M \bmod q$.
  2. $\psi_i = (g_{\mathsf{tag}_i^*}^{r_i}, g_{\mathsf{tag}_i^*}^{M_i} h_{\mathsf{tag}_i^*}^{r_i})$, for $i \in [k]$.

Security. We prove the security of $\mathsf{CS}^{\mathbf{F}} = (\mathsf{CS}^{\mathbf{F}}.\mathsf{Gen}, \mathsf{CS}^{\mathbf{F}}.\mathsf{Com}, \mathsf{CS}^{\mathbf{F}}.\mathsf{Ver})$ in the following lemma:

Lemma 1. *Let $\mathbf{F} = \{f_{\mathsf{tag}} : \mathbb{Z}_q \longrightarrow G_{\mathsf{tag}}\}_{\mathsf{tag} \in \{0,1\}^\lambda}$, where $f_{\mathsf{tag}}(x) := g^x_{\mathsf{tag}}$, such that the DDH is adaptively hard w.r.t. the collection of groups $\{(q, p_{\mathsf{tag}}, g_{\mathsf{tag}}, G_{\mathsf{tag}})\}_{\mathsf{tag} \in \{0,1\}^\lambda}$ and let $\mathbf{H} = \{\mathcal{H}_S\}_{S \subset \{0,1\}^\lambda}$ be the associated helper family defined in Fig. 1.*

Let $\{\mathsf{tag}^*_1, \ldots, \mathsf{tag}^*_k\} \subset \{0,1\}^\lambda$ be a tag collection. Then, the commitment scheme $\mathsf{CS}^{\mathbf{F}} = (\mathsf{CS}^{\mathbf{F}}.\mathsf{Gen}, \mathsf{CS}^{\mathbf{F}}.\mathsf{Com}, \mathsf{CS}^{\mathbf{F}}.\mathsf{Ver})$ w.r.t. $\{\mathsf{tag}^*_1, \ldots, \mathsf{tag}^*_k\} \subset \{0,1\}^\lambda$ is (i) perfectly binding and (ii) computationally hiding against PPT adversaries with access to any member $\mathcal{H}_S$ of $\mathbf{H}$ s.t. $\{\mathsf{tag}^*_1, \ldots, \mathsf{tag}^*_k\} \not\subset S$.

Proof. The perfect binding property of $\mathsf{CS}^{\mathbf{F}}$ is straightforward by the perfect correctness of ElGamal and the secret sharing scheme, and argued at an information theoretic level independently of $\mathbf{H}$. To prove the computational hiding property of $\mathsf{CS}^{\mathbf{F}}$ against helper-aided PPT adversaries, we will use reduction to the adaptive DDH hardness of $G_{\mathsf{tag}^*_1}, \ldots, G_{\mathsf{tag}^*_k}$.

Namely, let $\mathcal{A}$ be a PPT adversary aided by $\mathcal{H}_S$ s.t. for some $i \in [k] : \mathsf{tag}^*_i \notin S$, against the hiding property of $\mathsf{CS}^{\mathbf{F}}$. Observe that $\mathcal{H}_S$ for this $\mathbf{F}$ provides discrete logarithms as preimage replies.

We construct an adversary $\mathcal{B}$ that by having access to the discrete logarithm oracle $\mathcal{DL}(\mathsf{tag}^*_i, \cdot, \cdot)$, manages to break the adaptive DDH hardness for $G_{\mathsf{tag}^*_i}$. In particular, $\mathcal{B}$ on input $(g_{\mathsf{tag}^*_i}, g^x_{\mathsf{tag}^*_i}, g^y_{\mathsf{tag}^*_i}, g^z_{\mathsf{tag}^*_i})$ (where z is either equal to xy or random) and hardcoded with q executes the following steps:

1. It sets $\mathsf{ck}_i := (q, p_{\mathsf{tag}^*_i}, g_{\mathsf{tag}^*_i}, h_{\mathsf{tag}^*_i} := g^x_{\mathsf{tag}^*_i})$.
2. For $j \in [k] \setminus \{i\}$, it chooses a distinct tag $\mathsf{tag}_j \neq \mathsf{tag}^*_i$ and a random value $t_j \xleftarrow{\$} \mathbb{Z}_{\mathsf{tag}_j}$. It sets $\mathsf{ck}_j := (q, p_{\mathsf{tag}_j}, g_{\mathsf{tag}_j}, h_{\mathsf{tag}_j} := g^{t_j}_{\mathsf{tag}_j})$.
3. It provides $\mathcal{A}$ with $\mathsf{ck} = (\mathsf{ck}_1, \ldots, \mathsf{ck}_k)$ and emulates the hiding game for CS.
4. For a query (tag, β) from $\mathcal{A}$ intended for $\mathcal{H}_S$, it emulates a response as follows: if $\mathsf{tag} \in S$, (hence, $\mathsf{tag} \neq \mathsf{tag}^*_i$), then it forwards the query to $\mathcal{DL}(\mathsf{tag}^*_i, \cdot, \cdot)$ and replies with the oracle's response α. If $\mathsf{tag} \notin S$ (hence, including the case $\mathsf{tag} = \mathsf{tag}^*_i$), then it returns $\perp$.
5. It receives the challenge pair (M^0, M^1) from $\mathcal{A}$ and replies with an emulated commitment $\mathbf{c}^b \leftarrow \mathsf{Com}_{\mathsf{ck}}(M^b)$ as follows:
   (a) It chooses a random $b \in \{0,1\}$ and splits M^b into k shares $M^b_1, \ldots, M^b_k$.
   (b) For $j \in [k] \setminus \{i\}$, it computes a normal ElGamal encryption of M^b_j, ψ^b_j.
   (c) For $j = i$, it sets $\psi^b_i := (g^y_{\mathsf{tag}^*_i}, g^{M^b_i} g^z_{\mathsf{tag}^*_i})$.
   (d) It sets $\mathbf{c}^b := (\psi^b_1, \ldots \psi^b_k)$.
6. If $\mathcal{A}$'s guess matches b, then it outputs 1. Otherwise, it outputs 0.

First, note that by the information theoretic security of the secret sharing scheme, even though $\mathcal{A}$, by querying $\mathcal{H}_S$, can obtain the exponent t_j and thus totally recover some of (or even all) the shares M^b_j for $j \in [k] \setminus \{i\}$, it does not gain any information about M^b_i by these queries. Thus, the guess of $\mathcal{A}$ for M^b is independent given any collection of shares that does not include M^b_i.

The proof is completed similarly to the reduction of ElGamal IND-CPA security to the DHH assumption. Namely, let $\frac{1}{2} + \delta$ be the advantage of $\mathcal{A}$ in breaking the hiding property of $\mathsf{CS}^{\mathbf{F}}$. Next, assume that $\mathcal{B}$ gets a valid DDH tuple $(g_{\mathsf{tag}_i^*}, g^x_{\mathsf{tag}_i^*}, g^y_{\mathsf{tag}_i^*}, g^{xy}_{\mathsf{tag}_i^*})$, i.e. $z = xy$. In this case, $\mathcal{B}$ sets ψ_i^b so that it perfectly emulates the hiding game. On the other hand, if z is totally random, then the value $\psi_i^b := (g^y_{\mathsf{tag}_i^*}, g^{M_i^b} g^z_{\mathsf{tag}_i^*})$ is randomly distributed, independently from b.

We conclude that

$$\Big| \Pr\big[x, y \xleftarrow{\$} \mathbb{Z}_q : \mathcal{B}(g_{\mathsf{tag}_i^*}, g^x_{\mathsf{tag}_i^*}, g^y_{\mathsf{tag}_i^*}, g^{xy}_{\mathsf{tag}_i^*}) = 1\big] -$$
$$\Pr\big[x, y, z \xleftarrow{\$} \mathbb{Z}_q : \mathcal{B}(g_{\mathsf{tag}_i^*}, g^x_{\mathsf{tag}_i^*}, g^y_{\mathsf{tag}_i^*}, g^z_{\mathsf{tag}_i^*}) = 1\big] \Big| =$$
$$= \Big| \big(\frac{1}{2} + \delta\big) - \frac{1}{2} \Big| = \delta \ .$$

Namely, the distinguishing advantage of $\mathcal{B}$ is also δ. □

C.3 The Lapidot-Shamir (LS) fully input-delayed protocol

The LS protocol is a fully input-delayed protocol, as defined in Definition 5, i.e. both the prover and the simulator only need to know the size of the statement in order to produce the first move, while the actual statement is only needed for the third move. The LS protocol is a proof of knowledge of a Hamiltonian cycle HC of a given graph $G = (V, E)$ with size $|G|$. Consistently with the syntax of a Σ-protocol in Section 2, an LS proof consists of (1) the LS prover pair of algorithms $(a, \mathsf{st}_P) \leftarrow \mathsf{LS.Prv}_1(|G|)$ and $z \leftarrow \mathsf{LS.Prv}_2(\mathsf{st}_P, r, G, HC)$, (2) the LS verifier algorithm $\mathsf{LS}.V(a, G)$ that outputs a single bit challenge $r \in \{0, 1\}$, and (3) the verification algorithm $\mathsf{LS.Verify}(G, a, r, z)$ that outputs 0/1.

Description. More concretely, the protocol works as follows:

- $\mathsf{LS.Prv}_1(|G|)$: In the first move, the prover picks a random cycle RC with $|V|$ vertices and commits to RC in terms of a $|V| \times |V|$ adjacency matrix in an element-wise function. Then, it utilizes a perfectly binding commitment scheme to output the commitment to the adjacency matrix of RC as the first message a and a state $\mathsf{st}_P \in \{0, 1\}^*$.
- $\mathsf{LS}.V(a, G)$: The verifier sends a single bit challenge $r \in \{0, 1\}$.
- $\mathsf{LS.Prv}_2(\mathsf{st}_P, r, G, HC)$: In the third move, the prover takes as input the state $\mathsf{st}_P \in \{0, 1\}^*$, the challenge $r \in \{0, 1\}$, the statement $G = (V, E)$ and the corresponding witness (Hamiltonian cycle) HC. If $r = 0$, then it opens the committed RC outputs and sends them as the response z; otherwise, it finds a permutation π that maps the vertices of G to the adjacency matrix of RC such that RC is the Hamiltonian cycle HC, using a (perfectly binding) commitment scheme. It then opens all the committed adjacency matrix RC elements that correspond to non-edges of G. It outputs π and the opening to the adjacency matrix as the response z.

- $\mathsf{LS.Verify}(G, a, r, z)$: it takes as input the statement x and the transcript (a, r, z), and it outputs $0/1$, if the selected openings of the commitments to RC are consistent.

Security. The LS protocol satisfies input-delayed simulation, because there exists a simulator $\mathsf{LS.Sim} = (\mathsf{LS.Sim}_1, \mathsf{LS.Sim}_2)$ s.t. for any challenge r, $\mathsf{LS.Sim}$ can simulate a transcript that is computationally indistinguishable from the real one in an input-delayed manner. $\mathsf{LS.Sim} = (\mathsf{LS.Sim}_1, \mathsf{LS.Sim}_2)$ is defined as follows:

- $\mathsf{LS.Sim}_1(r, |G|)$: the simulator's output depends on the challenge r as follows:
  - If $r = 0$, then it acts as $\mathsf{LS.Prv}_1$. Namely, it picks a random cycle RC and outputs a commitment to the adjacency matrix of RC as a simulated first message a.
  - If $r = 1$, then it picks a random cycle RC, but now a is a commitment of an all-zero $|V| \times |V|$ matrix.

  In both cases, t he simulator state $\mathsf{st}_{\mathsf{Sim}}$ contains a, r and all the decommitment information for the committed matrix.
- $\mathsf{LS.Sim}_2(\mathsf{st}_{\mathsf{Sim}}, G)$:
  - If $r = 0$, then it acts as $\mathsf{LS.Prv}_2$. I.e., it outputs z as the opening to RC.
  - If $r = 1$, then it outputs z as a pair of (i) the permutation that maps RC to the vertices of G (in a fixed order), and (ii) the openings to the committed all-zero matrix that correspond to non-edges of G.

Note for $r = 0$, $\mathsf{LS.Sim}$ acts a real prover therefore it produces transcripts that are identically distributed to the real ones. For $r = 1$, the first message a and the permutation included in z are identically distributed to the ones in a real transcript, since RC is randomly chosen. Hence, sHVZK relies on the hiding property of the commitments between the real and the simulated adjacency matrix for RC.

Given Lemma 1 and the description of $\mathsf{LS.Sim} = (\mathsf{LS.Sim}_1, \mathsf{LS.Sim}_2)$, we prove the existence of a fully input-delayed Σ-protocol for any **NP** language that is secure against helper-aided adversaries.

Proposition 1. *Let $\mathbf{F} = \{f_{\mathsf{tag}} : \mathbb{Z}_{q_{\mathsf{tag}}} \longrightarrow G_{\mathsf{tag}}\}_{\mathsf{tag} \in \{0,1\}^\lambda}$ such that the DDH problem is adaptively hard in $\mathbf{F}$, and let $\mathbf{H} = \{\mathcal{H}_S\}_{S \subset \{0,1\}^\lambda}$ be the associated helper family defined in Fig. 1.*

Let $\{\mathsf{tag}_1^*, \ldots, \mathsf{tag}_k^*\} \subset \{0,1\}^\lambda$ be a tag collection. Then, for every $\mathcal{L} \in \mathbf{NP}$, there exists a fully input-delayed Σ-protocol for $\mathcal{L}$ that achieves (i) (perfect) completeness, (ii) special soundness and (iii) sHVZK against PPT adversaries with access to any member $\mathcal{H}_S$ of $\mathbf{H}$ such that $\{\mathsf{tag}_1^*, \ldots, \mathsf{tag}_k^*\} \not\subset S$.

Proof. Let $\mathsf{CS}^{\mathbf{F}} = (\mathsf{CS.Gen}^{\mathbf{F}}, \mathsf{CS.Com}^{\mathbf{F}}, \mathsf{CS.Ver}^{\mathbf{F}})$ be the commitment scheme presented in C.2. By the **NP**-completeness of the Hamiltonian Cycle problem, it suffices to show that the LS protocol over $\mathsf{CS}^{\mathbf{F}}$ satisfies the properties in Proposition 1. The proof is as in the standard proof the LS protocol, except from sHVZK where reference to helper access is necessary.

Perfect completeness: follows directly from the correctness of $\mathsf{CS.Gen}^{\mathbf{F}}$ that in turn, is based on the correctness of ElGamal encryption.

Special soundness: follows from the perfect binding property of $\mathsf{CS}^{\mathbf{F}}$. Namely, given two accepting transcripts $(G, a, 0, z_0)$ and $(G, a, 1, z_1)$ allows the extraction of the witness HC by $\mathsf{LS.Ext}$. This is true since for $r = 0$ one gets the random cycle RC and for $r = 1$ one gets the permutation that maps the random cycle in the actual cycle HC. Note that a single run of the LS proof has soundness error 1/2 and needs to be repeated λ times for soundness amplification to $2^{-\lambda}$.

sHVZK: let S such that $\{\mathsf{tag}_1^*, \ldots, \mathsf{tag}_k^*\} \not\subset S$ and $\mathcal{A}$ be a PPT adversary with access to $\mathcal{H}_S$ that breaks the sHVZK property of LS. Since for challenge $r = 0$, the simulator $\mathsf{LS.Sim}$ acts exactly as the actual prover, we have that for challenge $r = 1$, $\mathcal{A}$ can distinguish a simulated transcript from a real one on challenge 1 with some non-negligible advantage $\alpha(\lambda)$. W.l.o.g. (otherwise we can flip the output of $\mathcal{A}$), assume that

$$\begin{aligned}&\Pr\left[(a, \mathsf{st}_P) \leftarrow \mathsf{LS.Prv}_1(|G|); z \leftarrow \mathsf{LS.Prv}_2(\mathsf{st}_P, 1, G, HC) : \mathcal{A}(G, a, 1, z) = 1\right] \geq \\ &\geq \Pr\left[(a, \mathsf{st}_{\mathsf{Sim}}) \leftarrow \mathsf{LS.Sim}_1(1, |G|); z \leftarrow \mathsf{LS.Sim}_2(\mathsf{st}_{\mathsf{Sim}}, G) : \mathcal{A}(G, a, 1, z) = 1\right] + \alpha(\lambda)\ .\end{aligned} \tag{8}$$

We construct a PPT adversary $\mathcal{B}$ with access to $\mathcal{H}_S$ against the computational hiding property of $\mathsf{CS}^{\mathbf{F}}$. Namely, on input a commitment key ck, $\mathcal{B}$ operates as follows:

1. It specifies a graph G with vertex set V and Hamiltonian cycle HC w.r.t. the security parameter λ implied by ck.
2. It provides two multi-challenge vectors $M_0 = (0, \ldots, 0)$ and $M_1 = (1, \ldots, 1)$ consisting of $|V|$ elements. By a standard hybrid argument, we can refer to multi-challege hiding, as it is equivalent to single-challenge hiding up to a negligible error.
3. It replies to every query of $\mathcal{A}$ for $\mathcal{H}_S$, simply by forwarding the query and returning $\mathcal{H}_S$'s response.
4. It receives a vector of $|V|$ commitments to challenge bit b. Then, it runs an execution of LS as the prover does for challenge $r = 1$ with the following modification: $\mathcal{B}$ commits to the adjacency matrix of RC by plugging in the $|V|$ commitments to b in the matrix entries that correspond to the edges of RC.
5. It returns the response of $\mathcal{A}$.

Observe that when $b = 1$, then $\mathcal{B}$ interacts as a real prover, whereas when $b = 0$ it outputs a transcript identically distributed to the simulated ones. In both cases, this is on condition the challenge is 1. Thus, by Eq. (8), it holds that

$$
\begin{aligned}
&\Pr\left[\mathcal{B} \text{ wins}\right] = \\
=&\frac{1}{2}\cdot\Pr\left[\mathcal{B} \text{ wins} \mid b=1\right] + \frac{1}{2}\cdot\Pr\left[\mathcal{B} \text{ wins} \mid b=0\right] = \\
=&\frac{1}{2}\cdot\Pr\left[\mathcal{B} \text{ outputs } 1 \mid b=1\right] + \frac{1}{2}\cdot\Pr\left[\mathcal{B} \text{ outputs } 0 \mid b=0\right] = \\
=&\frac{1}{2} + \frac{1}{2}\cdot\Big(\Pr\left[\mathcal{B} \text{ outputs } 1 \mid b=1\right] - \Pr\left[\mathcal{B} \text{ outputs } 1 \mid b=0\right]\Big) = \\
=&\frac{1}{2} + \frac{1}{2}\cdot\Big(\Pr\left[\mathcal{A} \text{ outputs } 1 \mid b=1\right] - \Pr\left[\mathcal{A} \text{ outputs } 1 \mid b=0\right]\Big) \geq \\
\geq&\frac{1}{2} + \frac{\alpha(\lambda)}{2}\,.
\end{aligned}
$$

Therefore, $\mathcal{B}$ wins with non-negligible distinguishing advantage which contradicts to the hiding property of $\mathsf{CS}^{\mathbf{F}}$ proven in Lemma 1. This completes the proof. □

C.4 Proof of Theorem 2

Proof. **Crowd-verifiable Completeness.** In the following argumentation, we assume that the von Neumann sampling process for the computation of the coalescence outputs will not fail. By Theorem 1, this is something that happens with $1 - \mathsf{negl}(\lambda)$ probability, so we may include the $\mathsf{negl}(\lambda)$ failure error in the total error bound at the end of the completeness proof.

For a statement $x \in \mathcal{L} \cap \{0,1\}^{\mathrm{poly}(\lambda)}$ and a witness w for x, the completeness of $\Sigma.\Pi$ gives us that

$$
\begin{aligned}
&\Pr[(a, \mathsf{st}) \leftarrow \Sigma.\mathsf{Prv}_1(x,w); e \leftarrow \Sigma.V(x,a); r \leftarrow \Sigma.\mathsf{Prv}_2(x,w,e,\mathsf{st}) : \\
&\quad \Sigma.\mathsf{Verify}(x,a,e,z) = 0] \leq \delta(\lambda).
\end{aligned}
$$

Let $\mathcal{A}$ be a (not necessarily PPT) adversary against the completeness of the CVZK protocol $\mathsf{CVZK}.\Pi$ that may tamper at most t_1 verifier challenges, i.e. it can flip at most t_1 bits of the random challenge $C = (c_1, \ldots, c_n)$. By engaging in the completeness experiment $\mathbf{Expt}^{\mathsf{CVCompl}}_{(t_1,\mathcal{A},\mathcal{I}_{\mathsf{corr}})}(1^\lambda, x, w)$ (as defined in Def. 3), the adversary will win only if it manages to output an adversarial challenge $\hat{C}$ s.t. a non-accepting CVZK transcript $(A, \hat{C}, Z)$ is generated.

In the third round of $\mathsf{CVZK}.\Pi$, the prover algorithm $\mathsf{CVZK}.\mathsf{Prv}_2(x, w, A, \hat{C}, \mathsf{st}_P)$ will generate the n-bit string $E = \hat{C} \oplus R$, where R is a truly random n-bit string produced by $\mathsf{CVZK}.\mathsf{Prv}_1(x, w)$ and included in its state st_P. So, $E := (e_1, \ldots, e_{\log\lambda})$ is uniformly distributed. Note that this holds *unconditionally*, i.e., regardless of $\mathcal{A}$'s strategy.

Therefore, the strings $e_1, \ldots, e_{\log\lambda}$ that will be used as challenges for the $\log\lambda$ independent executions of $\mathsf{CVZK}.\Pi$ are uniformly random. By the completeness and uniformity of the challenge of $\varSigma.\varPi$ and the union bound, we have that

$$\begin{aligned}
&\Pr\Big[(a_i, \mathsf{st}_i) \leftarrow \varSigma.\mathsf{Prv}_1(x, w), i \in [\log\lambda]; (e_1, \ldots, e_{\log\lambda}) \leftarrow \mathbb{U}_n; \\
&\quad z_i \leftarrow \varSigma.\mathsf{Prv}_2(x, w, e_i, \mathsf{st}_i), i \in [\log\lambda] : \bigwedge_{i\in[\log\lambda]} \varSigma.\mathsf{Verify}(x, a_i, e_i, z_i) = 1\Big] \\
&\geq 1 - \sum_{i\in[\log\lambda]} \Pr\Big[(a_i, \mathsf{st}_i) \leftarrow \varSigma.\mathsf{Prv}_1(x, w); e_i \leftarrow \mathbb{U}_{n/\log\lambda}; \\
&\quad z_i \leftarrow \varSigma.\mathsf{Prv}_2(x, w, e_i, \mathsf{st}_i) : \varSigma.\mathsf{Verify}(x, a_i, e_i, z_i) = 0\Big] \geq \\
&\geq 1 - \delta(\lambda)\log\lambda\ .
\end{aligned} \tag{9}$$

To argue about the completeness of the input-delayed proofs, we first make use of the public samplability property of $\mathbf{F}$ (cf. property (1) of Definition 2) with error $\epsilon(\cdot)$. Namely, for every tag_ℓ, $\ell \in [n]$, it holds that

$$\Pr\left[\omega \leftarrow \mathbb{U}_{\lambda/\log^2\lambda} : \mathsf{IM}(\mathsf{tag}_\ell, \omega) \in Y_{\mathsf{tag}}\right] \geq 1 - \epsilon(\lambda)\ . \tag{10}$$

By Eq. (10), at least $(1 - \epsilon_\ell(\lambda))2^{\frac{\lambda}{\log^2\lambda}}$ strings in $\{0,1\}^{\frac{\lambda}{\log^2\lambda}}$ lead to the sampling of a valid statement for $\mathcal{L}^*_{\mathsf{tag}_\ell}$. We argue that by the construction of the coalescence $F(\cdot)$ over the strongly resilient function f_{res}, for every $\ell \in [n], j \in [\log\lambda]$, the output of $\mathsf{IM}(\mathsf{tag}_\ell, d_j)$ is a valid statement $x^*_{\ell,j} \in \mathcal{L}^*_{\mathsf{tag}_\ell}$.

Let $(G_1, \ldots, G_{\log\lambda})$ be the grouping of the bits of the input to the coalescence function that produce the output of $\lambda/\log^2\lambda$- bits strings $d_1, \ldots, d_{\log\lambda}$. Note that the adversary breaks completeness even if it manages to tamper the coalescence output of a single group so that it leads to the sampling of an invalid value (non-statement). Thus, it suffices to show that $\mathcal{A}$ has $\mathsf{negl}(\lambda)$ success probability, even if it concentrates all its tampering power of t_1 bits, upon a single group, say G_j. As in Theorem 1 proof, let $B_{j,1}, \ldots, B_{j,\lambda}$ denote the blocks inside the group G_j. $b_{j,1}, \ldots, b_{j,\lambda}$ be the bit sequence that, when given as input to the von Neumann rejection sampling process, produces d_j. Namely, $b_{j,k}$ is the output bit of $f_{\mathsf{res}}(B_{j,k})$.

Next, assume that $\mathcal{A}$ selects $t_{i,j}$ bits to corrupt for the block $B_{i,j}$, and $\forall i$: $\sum_{j=1}^{\lambda} t_{i,j} = t_1$. Let $\mathsf{Inf}_{i,j}$ be the event that over the input coins of $B_{i,j}$ the adversary $\mathcal{A}$ influences the output bit $b_{i,j} \leftarrow f_{\mathsf{res}}(B_{i,j})$, where $i \in [\log\lambda], j \in [\lambda]$. By the $(\varTheta(\frac{\log^2 m}{m}))$-strong resilience of f_{res}, if we set $m = \frac{n}{\lambda\log\lambda}$ (the input block length), then each corrupted bit in $B_{i,j}$ has probability $\varTheta(\frac{\log^3 n}{n^{1-\frac{1}{\gamma}}})$ to influence the output bit of $B_{i,j}$. By the union bound, the latter implies that

$$\Pr\left[\mathsf{Inf}_{i,j}\right] = \varTheta\Big(t_{i,j} \cdot \frac{\log^3 n}{n^{1-\frac{1}{\gamma}}}\Big)\ . \tag{11}$$

Now, let X be the random variable that counts the number of blocks from $B_{i,1}, \ldots, B_{i,\lambda}$ which output bit has been influenced. It is easy to see that X is the sum of λ Bernoulli trials with success probability $\Pr\left[\mathsf{Inf}_{i,1}\right], \ldots, \Pr\left[\mathsf{Inf}_{i,\lambda}\right]$, respectively. Thus, by Eq. (11), the mean of X is

$$\mu = \mathbb{E}[X] = \sum_{j=1}^{\lambda} \Theta\Big(t_{i,j} \cdot \frac{\log^3 n}{n^{1-\frac{1}{\gamma}}}\Big) = \Theta\Big(t_1 \cdot \frac{\log^3 n}{n^{1-\frac{1}{\gamma}}}\Big) \ .$$

By requiring that $t_1 \leq \frac{n^{1-\frac{1}{\gamma}}}{\log^2 n}$, we get that $\mu = \Theta(\log n)$. Thus, by the Chernoff bounds, we have that asymptotically

$$\Pr\left[X \geq \log^2 n\right] \leq \Pr\left[X \geq \mu \log^{1/2} n\right] \leq e^{-\Theta(\log^{3/2} n)} = \mathsf{negl}(\lambda) \ . \tag{12}$$

So with $1 - \mathsf{negl}(\lambda)$, $\mathcal{A}$ can not influence more than $\Theta(\log^2 n)$ out of the λ bits of the sequence $b_{j,1}, \ldots, b_{j,\lambda}$. Since to produce a bit of d_j, we need at least two input bits in the von Neumann sampling process, we deduce that $\mathcal{A}$ can neither influence more than $\Theta(\log^2 n)$ out of the $\lambda/\log^2 \lambda$ bits of d_j.

Recall that the $\lambda/\log^2 \lambda - \Theta(\log^2 n)$ uninfluenced bits of d_j are uniformly random. Therefore, given the adversarial tampering the output is randomly chose among a subset of $2^{\frac{\lambda}{\log^2 \lambda} - \Theta(\log^2 n)}$ strings.

In the worst case (hence, optimal in terms of adversarial strategy), these $2^{\frac{\lambda}{\log^2 \lambda} - \Theta(\log^2 n)}$ strings contain all the "bad" ones that lead to an invalid value (non-statement). By Eq. (10), the bad strings can be no more than $\epsilon(\lambda) 2^{\frac{\lambda}{\log^2 \lambda}}$ in total. Thus, the probability of randomly hitting a bad string is upper bounded by

$$\frac{\epsilon(\lambda) 2^{\frac{\lambda}{\log^2 \lambda}}}{2^{\frac{\lambda}{\log^2 \lambda} - \Theta(\log^2 n)}} = \epsilon(\lambda) 2^{\Theta(\log^2 n)} \ .$$

Recall that the above bound holds for every $\ell \in [n] \forall j \in [\log \lambda]$. Thus, by the union bound, we have that

$$\begin{aligned} &\Pr\,[\bigwedge_{\ell \in [n]} \bigwedge_{j \in [\log \lambda]} x^*_{\ell,j} = \mathsf{IM}(\mathsf{tag}_\ell, d_j) \in \mathcal{L}^*_{\mathsf{tag}_\ell}] \\ &\geq 1 - n \log \lambda \epsilon(\lambda) 2^{\Theta(\log^2 n)} \ . \end{aligned} \tag{13}$$

Given the validity of input-delayed statements, the completeness of all proofs for the validity of $x^*_{\ell,j}$, $\ell \in [n], j \in [\log \lambda]$ follows directly from the perfect completeness of the $\mathsf{InD}.\Pi$ proofs.

By Eq. (9) and (13) and including the $\mathsf{negl}(\lambda)$ failure error in von Neumann sampling, we conclude that for every $t_1 \leq \frac{n^{1-\frac{1}{\gamma}}}{\log^2 n}$, $\mathsf{CVZK}.\Pi$ has (t_1, ϵ_1)-completeness, where $\epsilon_1(\lambda) := \delta(\lambda) \log \lambda + n \log \lambda \epsilon(\lambda) 2^{\Theta(\log^2 n)} + \mathsf{negl}(\lambda)$.

Crowd-verifiable soundness. Given t_2-crowd verifiable validity that will be shown below, crowd verifiable soundness is easily deduced. Specifically, assume that there is a non-negligible function $\alpha(\cdot)$ and an (unbounded) adversary $\mathcal{A}$ s.t. for some $x \in \{0,1\}^{\mathsf{poly}(\lambda)}$

$$\Pr\left[\mathbf{Expt}^{\mathsf{CVSound}}_{(t_2,\mathcal{A},\mathcal{I}_{\mathsf{corr}})}(1^\lambda, x) = 1\right] \geq \alpha(\lambda)\ .$$

Then, by t_2-crowd verifiable validity, then we have that $x \in \mathcal{L}$ with non-negligible probability $\beta(\lambda)$, as the CVZK extractor utilizing $\mathcal{A}$ can output a witness for $x \in \mathcal{L}$ with non-negligible probability. Therefore for every $x \in \{0,1\}^{\mathsf{poly}(\lambda)} \setminus \mathcal{L}$ and every adversary $\mathcal{A}$ there is a negligible function $\epsilon_{x,\mathcal{A}}(\cdot)$ s.t.

$$\Pr\left[\mathbf{Expt}^{\mathsf{CVSound}}_{(t_2,\mathcal{A},\mathcal{I}_{\mathsf{corr}})}(1^\lambda, x) = 1\right] \leq \epsilon_{x,\mathcal{A}}(\lambda)\ .$$

Since for every λ, the set $\{\epsilon_{x,\mathcal{A}}(\lambda)\}_{x,\mathcal{A}}$ is lower bounded (e.g., by 0), by the supremum axiom of the real numbers we can define the function $\epsilon(\cdot)$

$$\epsilon_2(\lambda) := \inf(\{\epsilon_{x,\mathcal{A}}(\lambda)\}_{x,\mathcal{A}})\ .$$

Clearly, $\epsilon(\cdot)$ is negligible. Besides, by the definition of $\epsilon_{x,\mathcal{A}}(\cdot)$, we deduce that $\mathsf{CVZK}.\mathit{\Pi}$ achieves (t_2, ϵ_2)-crowd verifiable soundness.

Crowd-verifiable validity. Let $\mathcal{A}$ be an adversary against the crowd verifiable validity of $\mathsf{CVZK}.\mathit{\Pi}$ that acts as a malicious prover, corrupts up to t_2 verifiers and engages in $\mathbf{Expt}^{\mathsf{CVSound}}_{(t_2,\mathcal{A},\mathcal{I}_{\mathsf{corr}})}(1^\lambda)$. For some statement x, and subset $\mathcal{I}_{\mathsf{corr}} \subset [n]$ we define

$$p_x := \Pr\left[\mathbf{Expt}^{\mathsf{CVSound}}_{(t_2,\mathcal{A},\mathcal{I}_{\mathsf{corr}})}(1^\lambda, x) = 1\right]\ .$$

We construct a (candidate) PPT extractor algorithm $\mathsf{CVZK.Ext}$ for $\mathsf{CVZK}.\mathit{\Pi}$ that emulates $\mathbf{Expt}^{\mathsf{CVSound}}_{(t_2,\mathcal{A},\mathcal{I}_{\mathsf{corr}})}(1^\lambda, x)$ with standard rewinding and utilizes the extractor of $\mathit{\Sigma}.\mathit{\Pi}$, denoted by $\mathit{\Sigma}.\mathsf{Ext}$, to extract a witness for x, given the publicly samplable adaptive one-way function family $\mathbf{F}$. Then, we will analyze the success probability for $\mathsf{CVZK.Ext}$.

The algorithm $\mathsf{CVZK.Ext}$ given the code of $\mathcal{A}$, on input x and the set of indices of corrupted users $\mathcal{I}_{\mathsf{corr}}$, executes the following steps:

1. It starts $\mathbf{Expt}^{\mathsf{CVSound}}_{(t_2,\mathcal{A},\mathcal{I}_{\mathsf{corr}})}(1^\lambda, x)$, playing the role of honest verifiers.
2. It obtains $\big(A := (\{a_i\}_{i\in[\log\lambda]}, \{a^*_\ell\}_{\ell\in[n]})\big)$ from $\mathcal{A}(x, \mathcal{I}_{\mathsf{corr}})$.
3. **For** $\ell \in [n]$, it chooses a random bit c_ℓ to generate challenge $C := (c_1, \ldots, c_n) \in \{0,1\}^n$.
4. It obtains tampered challenge $\hat{C} := \langle \hat{c}_1, \ldots, \hat{c}_n \rangle$ and response $Z := (E, \{z_i\}_{i\in[\log\lambda]}, \{z^*_{\ell,j}\}^{j\in[\log\lambda]}_{\ell\in[n]})$ from $\mathcal{A}(x, \langle c_\ell \rangle_{\ell\in\mathcal{I}_{\mathsf{corr}}})$.
5. If $\mathsf{CVZK.Verify}(x, A, \hat{C}, Z) = 1$, then it rewinds $\mathcal{A}$ at the beginning of the challenge step providing $\mathcal{A}$ with a new challenge $\langle c'_\ell \rangle_{\ell\in\mathcal{I}_{\mathsf{corr}}}$ derived from a fresh challenge $C' := (c'_1, \ldots, c'_n) \in \{0,1\}^n$, where C' is computed as follows:

(a). Let $(G_1, \ldots, G_{\lambda\log\lambda})$ be the partition of C and $(G'_1, \ldots, G'_{\lambda\log\lambda})$ be the partition of C' into $\log\lambda$ groups of $\frac{n}{\log\lambda}$ bits.
(b). $\mathsf{CVZK.Ext}$ chooses randomly an index $i^* \in [\log\lambda]$.
(c). For every $i \in [\log\lambda] \setminus i^*$, it chooses G_i uniformly at random.

(d). It fixes $G'_{i^*} = G_{i^*}$.

If $\mathsf{CVZK.Verify}(x, A, \hat{C}, Z) = 0$, then $\mathsf{CVZK.Ext}$ aborts.

6. It obtains new tampered challenge $\hat{C}' := \langle \hat{c}'_1, \ldots, \hat{c}'_n \rangle$ and new response $Z' := (E', \{z'_i\}_{i \in [\log\lambda]}, \{{z'_{\ell,j}}^*\}_{\ell\in[n]}^{j\in[\log\lambda]})$ from $\mathcal{A}(x, \langle c'_1, \ldots, c'_n \rangle)$.
7. If $\mathsf{CVZK.Verify}(x, A, \hat{C}', Z') = 1$, then

(a). Let $F(\hat{C}) = (\hat{d}_1, \ldots, \hat{d}_{\log\lambda})$, and $F(\hat{C}') = (\hat{d}'_1, \ldots, \hat{d}'_{\log\lambda})$, be the coalescence images. Parse E, E' as $(e_1, \ldots, e_{\log\lambda})$ and $(e'_1, \ldots, e'_{\log\lambda})$, respectively.
(b). If there is an $i_0 \in [\log\lambda]$ s.t. $e_{i_0} \neq e'_{i_0}$, then it runs $\Sigma.\mathsf{Ext}$ on input the two valid transcripts $(x, a_{i_0}, e_{i_0}, z_{i_0})$ and $(x, a'_{i_0}, e'_{i_0}, z'_{i_0})$ and receives the output w of $\Sigma.\mathsf{Ext}$. Then, $\mathsf{CVZK.Ext}$ returns w as a valid witness for x.
(c). Otherwise, it holds that $(e_1, \ldots, e_{\log\lambda}) = (e'_1, \ldots, e'_{\log\lambda})$. Then, $\mathsf{CVZK.Ext}$ aborts.

If $\mathsf{CVZK.Verify}(x, A, \hat{C}', Z') = 0$, then $\mathsf{CVZK.Ext}$ aborts.

By the above description and the special soundness of $\Sigma.\Pi$, $\mathsf{CVZK.Ext}$ will certainly output a valid witness for x if the following events happen:

(i). $\mathsf{CVZK.Verify}(x, A, \hat{C}, Z) = \mathsf{CVZK.Verify}(x, A, \hat{C}', Z') = 1$, and
(ii). $E \neq E'$.

We lower bound the probability that these events happen via two claims. We begin by making use of the coalescence function $F(\cdot)$. Specifically, recall that Definition 4 refers to unbounded adversaries, thus it captures the state of $\mathcal{A}$ upon receiving a randomly chosen CVZK challenge C. Therefore, $(d_1, \ldots, d_{\log\lambda}) \leftarrow F(\hat{C})$ is a valid output for a run of the experiment $\mathbf{Expt}^{\mathsf{Coal}}_{(t,\mathcal{A},\mathcal{I}_{\mathsf{corr}})}(1^\lambda)$ in Definition 4. The latter implies that there exist events $\mathsf{G}_1, \ldots \mathsf{G}_m$ from the probability space of $\mathbf{Expt}^{\mathsf{CVSound}}_{(t_2,\mathcal{A},\mathcal{I}_{\mathsf{corr}})}(1^\lambda, x)$ such that for the following two conditions hold:

(1) $\Pr[\wedge_{i=1}^{\log\lambda} \neg \mathsf{G}_i] = \mathsf{negl}(\lambda)$, and

(2) for all $i \in [\log\lambda]$ and $\mathcal{I}_{\mathsf{corr}}$ output by $\mathcal{A}$, the random variable $(d_i | \mathsf{G}_i)$ is statistically $\mathsf{negl}(\lambda)$-close to $\mathbb{U}_{\lambda/\log^2\lambda}$.

By the construction of $F(\cdot)$ (cf. also proof of Theorem 1), each G_i denotes the event that $\mathcal{A}$ does not influence any of the λ bits output by the blocks of group G_i, where $(G_1, \ldots, G_{\log\lambda}) = C$. Thus, by condition (1) we have that

$$\Pr\left[\left(\mathbf{Expt}^{\mathsf{CVSound}}_{(t_2,\mathcal{A})}(1^\lambda, x) = 1\right) \wedge \left(\vee_{i=1}^{\log\lambda} \mathsf{G}_i\right)\right] = p_x - \mathsf{negl}(\lambda) \ . \tag{14}$$

By Eq. (14) and the union bound, we get that

$$\begin{aligned} p_x - \mathsf{negl}(\lambda) = &\Pr\left[\left(\mathbf{Expt}^{\mathsf{CVSound}}_{(t_2,\mathcal{A})}(1^\lambda, x) = 1\right) \wedge \left(\vee_{i=1}^{\log\lambda} \mathsf{G}_i\right)\right] = \\ = &\Pr\left[\vee_{i=1}^{\log\lambda} \left(\left(\mathbf{Expt}^{\mathsf{CVSound}}_{(t_2,\mathcal{A})}(1^\lambda, x) = 1\right) \wedge \mathsf{G}_i\right)\right] \leq \\ \leq &\sum_{i=1}^{\log\lambda} \left[\left(\mathbf{Expt}^{\mathsf{CVSound}}_{(t_2,\mathcal{A})}(1^\lambda, x) = 1\right) \wedge \mathsf{G}_i\right)\right] \ . \end{aligned} \tag{15}$$

By Eq. (15) and an averaging argument, we conclude that there exists an $i_0 \in [\log\lambda]$ and a negligible function $\delta(\cdot)$ s.t.

$$\Pr\left[\left(\mathbf{Expt}^{\mathsf{CVSound}}_{(t_2,\mathcal{A})}(1^\lambda, x) = 1\right) \wedge \mathsf{G}_{i_0}\right] \geq \frac{p_x}{\log\lambda} - \delta(\lambda)\ . \tag{16}$$

Coming back to the setting dictated by the description of $\mathsf{CVZK.Ext}$, we define as Uninf the event that in both runs (the first with challenge C and the rewinded run with challenge C' on condition that G_{i^*} is fixed) that $\mathsf{CVZK.Ext}$ performs the event G_{i^*}. We lower bound the probability $\mathsf{Valid} \wedge \mathsf{Uninf}$ happens in the following claim.

Claim 5 $\Pr[\mathsf{Valid} \wedge \mathsf{Uninf}] \geq \frac{(p_x)^3}{8(\log\lambda)^4} - \mathsf{negl}(\lambda)$.

Proof of claim: We prove the claim via a combinatorial argument along the lines of the splitting Lemma. Let $\mathbf{A}$ be the space of all first move messages A and $\mathbf{R}$ be the space of all response messages $(\hat{C}, Z)$ that $\mathcal{A}$ outputs during the execution of $\mathbf{Expt}^{\mathsf{CVSound}}_{(t_2,\mathcal{A})}(1^\lambda, x)$. For notation simplicity, the elements of $\mathbf{A}$, $\mathbf{R}$ are in one-to-one correspondence with the coins of $\mathcal{A}$, where w.l.o.g. we assume that $\mathcal{A}$ always outputs well-formed messages.

For the index i_0 guaranteed from Eq. (16), let $\mathsf{Good.Tr}_x$ be the set of triples $\left(A, C := (G_1, \ldots, G_{\log\lambda}), (\hat{C} := (\hat{G}_1, \ldots, \hat{G}_{\log\lambda}), Z)\right) \in \mathbf{A} \times \left(\{0,1\}^{\frac{n}{\log\lambda}}\right)^{\log\lambda} \times \mathbf{R}$ such that

1. $\mathsf{CVZK.Verify}(x, A, \hat{C}, Z) = 1$ (i.e. the transcript $(x, A, \hat{C}, Z)$ is accepting), and
2. the output d_{i_0} of G_{i_0} is not influenced (i.e. running f_{res} on the blocks G_{i_0} and $\hat{G}_{i_0}$ produces the same output d_{i_0}).

Let $\mathsf{Good.Run}_x$ be the set of pairs $(A, G) \in \mathbf{A} \times \{0,1\}^{\frac{n}{\log\lambda}}$ s.t.

$$\Big|\left\{(C, \hat{C}, Z) \mid \left(\left(A, C := (G_1, \ldots, G_{\log\lambda}), (\hat{C}, Z)\right) \in \mathsf{Good.Tr}_x\right) \wedge \right.$$
$$\left.\wedge \left(G_{i_0} = G\right)\right\}\Big| \geq \frac{1}{2} \cdot \left(\frac{p_x}{\log\lambda} - \delta(\lambda)\right) \cdot \left(2^{\frac{n}{\log\lambda}}\right)^{\log\lambda - 1} \cdot |\mathbf{R}|\ .$$

Observe that $\mathsf{CVZK.Ext}$ samples the honest verifiers slightly differently than in $\mathbf{Expt}^{\mathsf{CVSound}}_{(t_2,\mathcal{A},\mathcal{I}_{\mathsf{corr}})}(1^\lambda, x)$. Namely, it chooses n bits for all positions (including the ones in $\mathcal{I}_{\mathsf{corr}}$) and then provides $\mathcal{A}$ with only the honest verifiers' bits. It is straightforward that this way of sampling is (i) consistent with the coalescence experiment $\mathbf{Expt}^{\mathsf{Coal}}_{(t,\mathcal{A},\mathcal{I}_{\mathsf{corr}})}(1^\lambda)$ (cf. Definition 4), and (ii) that this leads to exactly the same distribution as choosing only the honest bits as in CVZK security experiments. Therefore, by Eq. (16) and the definition of $\mathsf{Good.Tr}_x$, $\mathsf{Good.Run}_x$, we have that

$$\begin{aligned}
&\left(\frac{p_x}{\log\lambda} - \delta(\lambda)\right) \cdot |\mathbf{A}| \cdot \left(2^{\frac{n}{\log\lambda}}\right)^{\log\lambda} \cdot |\mathbf{R}| \leq |\mathsf{Good.Tr}_x| = \\
&= \sum_{(A,G)} \left|\left\{(C,\hat{C},Z) \mid \left((A,C,(\hat{C},Z)) \in \mathsf{Good.Tr}_x\right) \wedge \left(G_{i_0} = G\right)\right\}\right| = \\
&= \sum_{(A,G)\in\mathsf{Good.Run}_x} \left|\left\{(C,\hat{C},Z) \mid \left((A,C,(\hat{C},Z)) \in \mathsf{Good.Tr}_x\right) \wedge \left(G_{i_0} = G\right)\right\}\right| + \\
&\quad + \sum_{(A,G)\notin\mathsf{Good.Run}_x} \left|\left\{(C,\hat{C},Z) \mid \left((A,C,(\hat{C},Z)) \in \mathsf{Good.Tr}_x\right) \wedge \left(G_{i_0} = G\right)\right\}\right| \Rightarrow \\
&\Rightarrow \left(\frac{p_x}{\log\lambda} - \delta(\lambda)\right) \cdot |\mathbf{A}| \cdot \left(2^{\frac{n}{\log\lambda}}\right)^{\log\lambda} \cdot |\mathbf{R}| \\
&\quad \leq |\mathsf{Good.Run}_x| \cdot \left(2^{\frac{n}{\log\lambda}}\right)^{\log\lambda - 1} \cdot |\mathbf{R}| + \\
&\quad + \left(|\mathbf{A}| \cdot 2^{\frac{n}{\log\lambda}} - |\mathsf{Good.Run}_x|\right)\frac{1}{2} \cdot \left(\frac{p_x}{\log\lambda} - \delta(\lambda)\right) \cdot \left(2^{\frac{n}{\log\lambda}}\right)^{\log\lambda - 1} \cdot |\mathbf{R}| \Rightarrow \\
&\Rightarrow \left(\frac{p_x}{\log\lambda} - \delta(\lambda)\right) \cdot |\mathbf{A}| \cdot \left(2^{\frac{n}{\log\lambda}}\right)^{\log\lambda} \cdot |\mathbf{R}| \\
&\quad \leq |\mathsf{Good.Run}_x| \cdot \left(2^{\frac{n}{\log\lambda}}\right)^{\log\lambda - 1} \cdot |\mathbf{R}| + \\
&\quad + \left(|\mathbf{A}| \cdot 2^{\frac{n}{\log\lambda}}\right)\frac{1}{2} \cdot \left(\frac{p_x}{\log\lambda} - \delta(\lambda)\right) \cdot \left(2^{\frac{n}{\log\lambda}}\right)^{\log\lambda - 1} \cdot |\mathbf{R}| \Rightarrow \\
&\Rightarrow |\mathsf{Good.Run}_x| \geq \frac{1}{2} \cdot \left(\frac{p_x}{\log\lambda} - \delta(\lambda)\right) \cdot |\mathbf{A}| \cdot 2^{\frac{n}{\log\lambda}} \cdot \log\lambda\,.
\end{aligned} \tag{17}$$

If in the first run, assume the pair $(A, C := (G_1, \ldots, G_{\log\lambda}))$ is s.t. for some i_0, $(A, G_{i_0}) \in \mathsf{Good.Run}_x$. Let Guess be the event that by randomly choosing i^* in Step 5(d), $\mathsf{CVZK.Ext}$ successfully guesses $i^* = i_0$. Then, by definition of $\mathsf{Good.Run}_x$, and the description of $\mathsf{CVZK.Ext}$ (Steps 5(a)-(d)), we have that

$$\Pr\left[\mathsf{Valid} \wedge \mathsf{Uninf} \;\middle|\; (A, G_{i_0}) \in \mathsf{Good.Run}_x, \mathsf{Guess}\right] \geq \frac{1}{4} \cdot \left(\frac{p_x}{\log\lambda} - \delta(\lambda)\right)^2 . \tag{18}$$

By Eq. (17) and (18), the negligibility of $\delta(\cdot)$, and the fact that $\Pr[\mathsf{Guess}] = \frac{1}{\log\lambda}$, where the guess is independently at random, we conclude that

$$\begin{aligned}
&\Pr[\mathsf{Valid} \wedge \mathsf{Uninf}] \\
&\geq \Pr\left[\mathsf{Valid} \wedge \mathsf{Uninf} \wedge \left((A, G_{i_0}) \in \mathsf{Good.Run}_x\right) \wedge \mathsf{Guess}\right] = \\
=&\Pr\left[\mathsf{Valid} \wedge \mathsf{Uninf} \;\middle|\; (A, G_{i_0}) \in \mathsf{Good.Run}_x, \mathsf{Guess}\right] \cdot \\
&\qquad \cdot \Pr\left[(A, G_{i_0}) \in \mathsf{Good.Run}_x\right] \cdot \Pr\left[\mathsf{Guess}\right] \geq \\
\geq&\frac{1}{4} \cdot \left(\frac{p_x}{\log\lambda} - \delta(\lambda)\right)^2 \cdot \frac{1}{2} \cdot \left(\frac{p_x}{\log\lambda} - \delta(\lambda)\right) \cdot \frac{1}{\log\lambda} \geq \\
\geq&\frac{(p_x)^3}{8(\log\lambda)^4} - \mathsf{negl}(\lambda)\,.
\end{aligned}$$

(End of proof of Claim) $\dashv$

Subsequently, in the following claim, we apply the properties of the coalescence function $F(\cdot)$ and the security of the publicly samplable adaptive OWF family $\mathbf{F}$ to prove that the probability that $\mathsf{Valid}, \mathsf{Uninf}$ and $E = E'$ happen is negligible.

Claim 6 $\Pr[\mathsf{Valid} \wedge \mathsf{Uninf} \wedge (E = E')] = \mathsf{negl}(\lambda)$.

Proof of claim: Assume for the sake of contradiction that there is a non-negligible function $\alpha(\cdot)$ s.t.

$$\Pr\left[\mathsf{Valid} \wedge \mathsf{Uninf} \wedge (E = E')\right] \geq \alpha(\lambda) \ .$$

We show that we can construct a PPT algorithm $\mathcal{B}$ that breaks the security publicly samplable adaptive one-way function family

$$\mathbf{F} = \{f_{\mathsf{tag}} : X_{\mathsf{tag}} \mapsto Y_{\mathsf{tag}}\}_{\mathsf{tag} \in \{0,1\}^{\frac{\lambda}{\log^2 \lambda}}}$$

with non-negligible probability $\alpha'(\cdot)$. Namely, there is a $\mathsf{tag} \in \{0,1\}^{\frac{\lambda}{\log^2 \lambda}}$ s.t.

$$\Pr\left[d \leftarrow U_{\lambda/\log^2 \lambda} : \mathcal{B}^{\mathcal{O}(\mathsf{tag},\cdot,\cdot)}\big(\mathsf{tag}, d\big) = f^{-1}_{\mathsf{tag}}\big(\mathsf{IM}(\mathsf{tag}, d)\big)\right] \geq \alpha'(\lambda) \ .$$

In particular, the algorithm $\mathcal{B}$ is hardcoded with tag and $(x, \mathcal{I}_{\mathsf{corr}})$ and is given the code of $\mathsf{CVZK.Ext}$, $\mathcal{A}$, the algorithm Sample guaranteed by the efficient samplability of $F(\cdot)$, and the knowledge extractor of $\mathsf{InD}.\mathit{\Pi}$, denoted by $\mathsf{InD.Ext}$ and guaranteed by the special soundness of $\mathsf{InD}.\mathit{\Pi}$. On input a randomly chosen value d, the algorithm $\mathcal{B}$ executes the following steps:

1. It randomly chooses $\ell_0 \in [n] \setminus \mathcal{I}_{\mathsf{corr}}$ setting $\mathsf{tag}_{\ell_0} \leftarrow \mathsf{tag}$.
2. It executes the steps of $\mathsf{CVZK.Ext}^{\mathcal{A}}(x, \mathcal{I}_{\mathsf{corr}})$ with the following modification for sampling C: for $i \in [\log \lambda], i \neq i^*$ (where as before, i^* is the guess of $\mathsf{CVZK.Ext}$) for fixing the group G_i, it randomly picks $d_i \leftarrow \{0,1\}^{\lambda/\log^2 \lambda}$; it then sets $d_{i^*} := d$ and invokes $\mathsf{Sample}(d_1, \ldots, d_{\log \lambda})$, obtaining $C \in \{0,1\}^n$. If $F(C) \neq (d_1, \ldots, d_{\log \lambda})$, then it aborts.
3. Let $\hat{C}$ and $\hat{C}'$ be the adversarially tampered challenges. If d_{i^*} and d'_{i^*} derived by $F(\hat{C})$ and $F(\hat{C}')$ are not equal, then $\mathcal{B}$ aborts. Note that if $d_{i^*} = d'_{i^*}$, then the adaptive OWF sampler on input d_{i^*} and any tag tag_ℓ, $\ell \in [n]$, outputs the same statement $x^*_{\ell,i^*} := \beta_{\ell,i^*} \leftarrow \mathsf{IM}(\mathsf{tag}_\ell, d_{i^*})$ in both runs.
4. Let $R := (r_1, \ldots, r_n) = E \oplus \hat{C}$ and $R' := (r'_1, \ldots, r'_n) = E' \oplus \hat{C}'$. If $r_{\ell_0} \neq r'_{\ell_0}$, then $\mathcal{B}$ aborts.
5. If $\mathsf{CVZK.Verify}(x, A, \hat{C}, Z) = \mathsf{CVZK.Verify}(x, A, \hat{C}', Z') = 1$ (i.e. Valid happens as well as $d_{i^*} = d'_{i^*}$, and $r_{\ell_0} \neq r'_{\ell_0}$), then $\mathcal{B}$

(a). Computes the OWF challenge $\beta \leftarrow \mathsf{IM}(\mathsf{tag}, d)$ (i.e. $\beta = x^*_{\ell_0,i^*}$).

(b). Runs $\mathsf{InD.Ext}$ on input the two valid transcripts $(\beta, a^*_{\ell_0,i^*}, r_{\ell_0}, z^*_{\ell_0,i^*})$ and $(\beta, a^*_{\ell_0,i^*}, r'_{\ell_0}, z^{*'}_{\ell_0,i^*})$ and receives the output α of $\mathsf{InD.Ext}$. Then, $\mathsf{CVZK.Ext}$ returns α as a preimage for β.

Clearly, by the special soundness of $\mathsf{InD}.\mathit{\Pi}$, the algorithm $\mathcal{B}$ always returns a valid preimage when the event $\mathsf{Valid} \wedge (d_{i^*} = d'_{i^*}) \wedge (r_{\ell_0} \neq r'_{\ell_0})$ happens. Recall that Uninf is the event where in both runs, the output corresponding to the fixed i^*-th group G_{i^*} is uninfluenced. Thus, when Uninf happens d'_{i^*} and d'_{i^*} are always equal, and specifically equal to $\mathcal{B}$'s input, d, which means that $\mathcal{B}$ breaks the adaptive one-wayness of $\mathbf{F}$.

We first argue that $\mathcal{B}$ emulates $\mathsf{CVZK.Ext}^{\mathcal{A}}(x, \mathcal{I}_{\mathsf{corr}})$ with $1 - \mathsf{negl}(\lambda)$ probability. The latter holds by the efficient samplability of the coalescence function $F(\cdot)$, as proven in Theorem 1. Namely, since the adaptive OWF challenge d provided to $\mathcal{B}$ is randomly chosen, the vector $d_1, \ldots, d_{\log\lambda}$ that $\mathcal{B}$ generates at step 2 is uniformly distributed over $\left(\{0,1\}^{\lambda/\log^2\lambda}\right)^{\log\lambda}$. Thus, with $1 - \mathsf{negl}(\lambda)$ probability $\mathsf{Sample}(d_1, \ldots, d_{\log\lambda})$ outputs a value C s.t. $F(C) = (d_1, \ldots, d_{\log\lambda})$, so $\mathcal{B}$ will not abort in this step. In addition, C is sampled statistically $\mathsf{negl}(\lambda)$-close to $\mathbb{U}_n$, i.e., according to the original CVZK sampling that $\mathsf{CVZK.Ext}$ follows.

Let $\mathsf{Ham}(\cdot,\cdot)$ denote the Hamming distance between two strings of equal length. Recall that $\hat{C}, \hat{C}'$ derived by the tampering of at most $t_2 \leq \frac{n^{1-\frac{1}{\gamma}}}{\log^3 n}$ bit positions in the two n-bit challenges $C = (G_1, \ldots, G_{\lambda\log\lambda})$ and $C' = (G'_1, \ldots, G'_{\lambda\log\lambda})$ respectively. Since C' is generated by fixing the bits in G'_{i^*} the same as G_{i^*} and randomly flipping all other positions, we have that C and C' differ at $\frac{1}{2}\left(n - \frac{n}{\log\lambda}\right)$ bit positions on average. Hence, by the Chernoff bounds we have that

$$\Pr\left[\mathsf{Ham}(C, C') \geq n/3\right] \geq 1 - e^{-\frac{(\frac{n}{2} - \frac{n}{2\log\lambda})(\frac{n}{6} + \frac{n}{2\log\lambda})^2}{2}} = 1 - \mathsf{negl}(\lambda)\ . \tag{19}$$

Now suppose that C, C' differ in at least $n/3$ bit positions. In this case, since $\frac{n^{1-\frac{1}{\gamma}}}{\log^3 n} = o(n)$ for any $\gamma > 0$ and sufficiently large n, we deduce that whatever is the tampering strategy of $\mathcal{A}$, it is impossible that $\mathcal{A}$ can produce two adversarial challenges $\hat{C}, \hat{C}'$ that are "close". Specifically, by Eq. (19) and the fact that $\mathcal{B}$ almost perfectly emulates $\mathsf{CVZK.Ext}^{\mathcal{A}}(x)$, we have that with $1 - \mathsf{negl}(\lambda)$ probability and for sufficiently large n,

$$\mathsf{Ham}(\hat{C}, \hat{C}') \geq \frac{n}{3} - 2t_2 \geq \frac{n}{3} - 2\frac{n^{1-\frac{1}{\gamma}}}{\log^3 n} \geq \frac{n}{4}\ . \tag{20}$$

Since $\mathcal{B}$ chooses ℓ_0 uniformly at random, by Eq. (20) we get that

$$\Pr\left[\hat{c}_{\ell_0} \neq \hat{c}'_{\ell_0}\right] \geq \frac{1}{4} - \mathsf{negl}(\lambda)\ . \tag{21}$$

By our hypothesis, the fact that $R = E \oplus \hat{C}$, $R' = E' \oplus \hat{C}'$ and Eq. (21), we have that

$$\begin{aligned}&\Pr\left[\mathsf{Valid}\wedge\mathsf{Uninf}\wedge(E=E')\wedge(r_{\ell_0}\neq r'_{\ell_0})\right]=\\=&\Pr\left[\mathsf{Valid}\wedge\mathsf{Uninf}\wedge(E=E')\right]\cdot\Pr\left[r_{\ell_0}\neq r'_{\ell_0}\;\middle|\;\mathsf{Valid}\wedge\mathsf{Uninf}\wedge(E=E')\right]=\\=&\Pr\left[\mathsf{Valid}\wedge\mathsf{Uninf}\wedge(E=E')\right]\cdot\Pr\left[\hat{c}_{\ell_0}\neq\hat{c}'_{\ell_0}\right]\geq\\\geq&\frac{\alpha(\lambda)}{4}-\mathsf{negl}(\lambda)\ .\end{aligned}\tag{22}$$

By the description of $\mathcal{B}$, Eq. (22) provides a non-negligible lower bound for the success probability of $\mathcal{B}$ in returning a valid preimage for the adaptive OWF challenge d. In detail, by setting $\alpha'(\lambda) := \frac{\alpha(\lambda)}{4} - \mathsf{negl}(\lambda)$, we conclude that

$$\Pr\left[d\leftarrow U_{\lambda/\log^2\lambda}:\mathcal{B}^{\mathcal{O}(\mathsf{tag},\cdot,\cdot)}\big(\mathsf{tag},d\big)=f^{-1}_{\mathsf{tag}}\big(\mathsf{IM}(\mathsf{tag},d)\big)\right]\geq\alpha'(\lambda)\ ,$$

which leads to contradiction, due to the security of $\mathbf{F}$. Thus, it must hold that

$$\Pr[\mathsf{Valid}\wedge\mathsf{Uninf}\wedge(E=E')]=\mathsf{negl}(\lambda)\ .$$

(End of proof of Claim) ⊣

Finally, the crowd verifiable validity of our construction follows directly by applying the Claims 5 and 6, where we get that

$$\begin{aligned}&\Pr\left[w^*\leftarrow\mathsf{CVZK.Ext}^{\mathcal{A}}(x,\mathcal{I}_{\mathsf{corr}}):(x,w^*)\in R_{\mathcal{L}}\right]\geq\\\geq&\Pr\left[\mathsf{Valid}\wedge(E\neq E')\right]\geq\Pr\left[\mathsf{Valid}\wedge\mathsf{Uninf}\wedge(E\neq E')\right]=\\=&\Pr\left[\mathsf{Valid}\wedge\mathsf{Uninf}\right]-\Pr\left[\mathsf{Valid}\wedge\mathsf{Uninf}\wedge(E\neq E')\right]\geq\\\geq&\frac{(p_x)^3}{8(\log\lambda)^4}-\mathsf{negl}(\lambda)\ .\end{aligned}$$

Thus, if p_x is non-negligible, then $\mathsf{CVZK.Ext}$ extracts a valid witness with non-negligible probability as well.

Crowd-verifiable ZK. For some $x, t_3, \mathcal{I}_{\mathsf{corr}}$, s.t. $|\mathcal{I}_{\mathsf{corr}}| \leq t_3$ let $\mathcal{A}$ be a PPT adversary against the crowd-verifiable zero-knowledge of the CVZK protocol $\mathsf{CVZK}.\Pi$ (presented in Section 4) that can tamper at most t_3 verifier challenges. Namely, the adversary will win if it manages to distinguish the execution of the experiments $\mathbf{Expt}^{\mathsf{CVZK}}_{(\mathsf{Ideal},\mathcal{A},\mathcal{I}_{\mathsf{corr}})}(1^\lambda, x)$ and $\mathbf{Expt}^{\mathsf{CVZK}}_{(\mathsf{Real},\mathcal{A},\mathcal{I}_{\mathsf{corr}})}(1^\lambda, x)$ (as defined in Def. 3) for $x \in \mathcal{L}$.

Assume that $\mathcal{A}$ has access to a helper $\mathcal{H}_S$ from $\mathbf{H}$, where (i) $\{\mathsf{tag}_\ell\}_{\ell\in\mathcal{I}_{\mathsf{corr}}} \subseteq S$ and (ii) $\{\mathsf{tag}_\ell\}_{\ell\in[n]\setminus\mathcal{I}_{\mathsf{corr}}} \cap S = \emptyset$. In addition, assume that $\Sigma.\Pi$ and $\mathsf{InD}.\Pi$ are sHVZK against PPT distinguishers that have access to the said helper $\mathcal{H}_S$. We start by explaining how to construct the CVZK simulator pair $\mathsf{CVZK.Sim} = (\mathsf{CVZK.Sim}_1, \mathsf{CVZK.Sim}_2^{\mathcal{H}_S})$ that will be used in $\mathbf{Expt}^{\mathsf{CVZK}}_{(\mathsf{Ideal},\mathcal{A},\mathcal{I}_{\mathsf{corr}})}(1^\lambda, x)$.

Recall that our CVZK construction works over any Σ-protocol and utilizes a fully input-delayed Σ-protocol so that the prover is convincing if it either knows a witness for the underlying Σ-protocol or knows a witness for the underlying

$\mathsf{InD}.\mathit{\Sigma}$-protocol. However, the witness for the $\mathsf{InD.Sim}$ proof is never known as it depends on statements generated using the verifiers' random challenges/bits that are received in the second step of the CVZK protocol (thus the simulator from $\mathsf{InD}.\mathit{\Pi}$ is used in the main CVZK construction).

In the construction of the simulator there are two key ideas:

1. Since the witness w of the underlying $\mathit{\Sigma}$-protocol is no longer known, we replace the $\mathit{\Sigma}$-protocol prover by its corresponding simulator $\mathit{\Sigma}.\mathsf{Sim}$. Thus, in $\mathsf{CVZK.Sim}_1$ we now use $\mathit{\Sigma}.\mathsf{Sim}$ which when given the $\mathit{\Sigma}$-protocol challenges from $\mathsf{CVZK.Sim}_1$ it simulates the full transcript of the $\mathit{\Sigma}$-protocol which is stored in the state of the CVZK simulator $\mathsf{st}_{\mathsf{Sim}}$.
2. The input-delayed proof is relevant to the verifiers' random coins. Since in the ZK definition we allow for corruption of verifiers we will treat the input-delayed part of the CVZK simulator in a separate way for corrupted and non-corrupted verifiers. Namely, $\mathsf{CVZK.Sim}$ is identical to $\mathsf{CVZK.Prv}$ for all $\ell \notin \mathcal{I}_{\mathsf{corr}}$, however it is no longer possible to use the $\mathsf{InD.Sim}$ simulator for the corrupted verifiers. The reason is that the corrupted verifiers are now allowed to flip their coins in the duration of the protocol (changing C to C' between executions of $\mathsf{CVZK.Sim}_1$ and $\mathsf{CVZK.Sim}_2$). As a result, $\mathsf{InD.Sim}$ cannot predict which coins to use in advance. To overcome this problem, for all $\ell \in \mathcal{I}_{\mathsf{corr}}$ we use the actual $\mathsf{InD}.\mathit{\Pi}$ prover in the first step of the CVZK simulator, $\mathsf{CVZK.Prv}_1$, which is fine since in input-delayed proofs the statement does not have to be known in advance. Then, once the coins of all verifiers are received (corrupted or not) and the statements are set, we use the helper $\mathcal{H}_S$, as described in Fig. 1, to extract the witnesses (preimages) for the statements corresponding to the corrupted verifiers and conclude the input-delayed $\mathit{\Sigma}$-proof using the actual prover. Recall that $\{\mathsf{tag}_\ell\}_{\ell\in\mathcal{I}_{\mathsf{corr}}} \subseteq S$, so $\mathcal{H}_S$ allows the extraction of witness for all statements corresponding to the corrupted verifiers.

Formally, $\mathsf{CVZK.Sim} = (\mathsf{CVZK.Sim}_1,\ \mathsf{CVZK.Sim}_2^{\mathcal{H}_S})$ runs as follows:

1. $\mathsf{CVZK.Sim}_1(x, \mathcal{I}_{\mathsf{corr}})$: (where $\mathcal{I}_{\mathsf{corr}}$ is the set of corrupted verifiers.)
   - Pick random $E \leftarrow \{0,1\}^n$, and parse $E = (e_1, \ldots, e_{\log\lambda})$.
   - For $\ell \notin \mathcal{I}_{\mathsf{corr}}$, choose a random bit c_ℓ.
   - For $\ell \notin \mathcal{I}_{\mathsf{corr}}$, set r_ℓ as the XOR of c_ℓ and the ℓ-th bit of E.
   - For $i \in [\log\lambda]$, run $(a_i, z_i) \leftarrow \mathit{\Sigma}.\mathsf{Sim}(x, e_i)$.
   - For $\ell \in [n]$:
     - If $\ell \notin \mathcal{I}_{\mathsf{corr}}$: run $(a^*_\ell, \mathsf{st}^*_\ell) \leftarrow \mathsf{InD.Sim}_1(r_\ell, \mathit{size})$.
     - If $\ell \in \mathcal{I}_{\mathsf{corr}}$: for all $j \in [\log\lambda]$ run $(a^*_{\ell,j}, \mathsf{st}^*_{\ell,j}) \leftarrow \mathsf{InD}.\mathit{\Sigma}.\mathsf{Prv}_1(\mathit{size})$.
   - Output $A := (\{a_i\}_{i\in[\log\lambda]}, \{a^*_{\ell,j}\}^{j\in[\log\lambda]}_{\ell\in[n]}), \langle c_\ell\rangle_{\ell\in[n]\setminus\mathcal{I}_{\mathsf{corr}}}$,

     and $\mathsf{st}_{\mathsf{Sim}} := (E, \{z_i\}_{i\in[\log\lambda]}, \{\mathsf{st}^*_{\ell,j}\}^{j\in[\log\lambda]}_{\ell\in[n]})$.
2. $\mathsf{CVZK.Sim}_2^{\mathcal{H}_S}(\mathsf{st}_{\mathsf{Sim}}, C', \mathcal{I}_{\mathsf{corr}})$: (where C' can differ from C only for the indices in $\mathcal{I}_{\mathsf{corr}}$.)
   - Parse $\mathsf{st}_{\mathsf{Sim}} = (E, \{z_i\}_{i\in[\log\lambda]}, \{\mathsf{st}^*_\ell\}_{\ell\in[n]})$.
   - Set $R' := (r'_1, \ldots, r'_n) := E \oplus C'$.

- Compute $(d_1, \ldots, d_{\log\lambda}) \leftarrow F(C')$
- Compute $(e_1, \ldots, e_{\log\lambda}) \leftarrow E$.
- For $\ell \in [n]$ and for $j \in [\log\lambda]$:
  - Run $\beta_{\ell,j} \leftarrow \mathsf{IM}(\mathsf{tag}_\ell, d_j)$ and define the statement $x^*_{\ell,j} = \beta_{\ell,j}$.
  - If $\ell \notin \mathcal{I}_{\mathsf{corr}}$: $z^*_{\ell,j} \leftarrow \mathsf{InD.Sim}_2(\mathsf{st}^*_{\ell,j}, x^*_{\ell,j})$. Note that for $\ell \notin \mathcal{I}_{\mathsf{corr}}$, it holds that $r'_\ell = r_\ell$, therefore $\mathsf{InD.Sim}_2$ can complete the proof consistently.
  - If $\ell \in \mathcal{I}_{\mathsf{corr}}$: make query $(\mathsf{tag}_\ell, \beta_{\ell,j})$ to $\mathcal{H}_S$ to obtain an $\alpha_{\ell,j}$ such that $f_{\mathsf{tag}_\ell}(\alpha_{\ell,j}) = \beta_{\ell,j}$. Then, complete the proof by running $z^*_\ell \leftarrow \mathsf{InD.Prv}_2(x^*_{\ell,j}, \alpha_{\ell,j}, r'_\ell, x^*_{\ell,j}, \mathsf{st}^*_{\ell,j})$.
- Output $Z := (E, \{z_i\}_{i\in[\log\lambda]}, \{z^*_{\ell,j}\}^{j\in[\log\lambda]}_{\ell\in[n]})$.

We will now show that the view generated by $\mathsf{CVZK.Sim}$ is computationally indistinguishable from the view of the adversary $\mathcal{A}$ in an execution with the honest prover $\mathsf{CVZK.Prv}$ when $\mathsf{CVZK.Sim}$ is equipped with the helper oracle $\mathcal{H}_S$. Our proof is done through a sequence of hybrid experiments defined below.

Hybrid 0: The view of $\mathcal{A}$ is (x, A, C, Z) and is generated by $\mathsf{CVZK.Sim}$, as in the ideal experiment $\mathbf{Expt}^{\mathsf{CVZK}}_{(\mathsf{Ideal}, t_3, \mathcal{A}, \mathcal{I}_{\mathsf{corr}})}(1^\lambda, x)$.

Hybrid 1: In this hybrid experiment, we consider an alternative CVZK simulator $\mathsf{CVZK.Sim}' = (\mathsf{CVZK.Sim}'_1,\ \mathsf{CVZK.Sim}'^{\mathcal{H}_S}_2)$ that is given the witness w. Thus, for all $i \in [\log\lambda]$, $\mathsf{CVZK.Sim}'_1$ generates a_i's using the actual Σ-protocol prover $\Sigma.\mathsf{Prv}_1(x, w)$ and $\mathsf{CVZK.Sim}'_2$ generates z_i's using $\Sigma.\mathsf{Prv}_2(\mathsf{st}_i, e_i)$.

Hybrid 2: This final hybrid is the real game $\mathbf{Expt}^{\mathsf{CVZK}}_{(\mathsf{Real}, t_3, \mathcal{A}, \mathcal{I}_{\mathsf{corr}})}(1^\lambda, x, w)$, i.e. the view of $\mathcal{A}$ is generated by interacting with the actual prover $\mathsf{CVZK.Prv}$.

– *Indistinguishability between Hybrid 1 and Hybrid 0*: By the sHVZK property against PPT distinguishers with access to $\mathcal{H}_S$ of the underlying Σ-protocol $\Sigma.\Pi$, Hybrid 1 is computationally indistinguishable from Hybrid 0. Note that most common Σ-protocols achieve at least statistical (if not perfect) sHVZK, thus computational sHVZK w.r.t. $\mathcal{H}_S$ is implied. Namely, after the execution of the updated $\mathsf{CVZK.Sim}'_1$, the view for $\mathcal{A}$ will be $A := (\{a_i\}_{i\in[\log\lambda]}, \{a^*_{\ell,j}\}^{j\in[\log\lambda]}_{\ell\in[n]})$, $\langle c_\ell\rangle_{\ell\in[n]\setminus\mathcal{I}_{\mathsf{corr}}}$ where $\{a^*_{\ell,j}\}^{j\in[\log\lambda]}_{\ell\in[n]}$ and $\langle c_\ell\rangle_{\ell\in[n]\setminus\mathcal{I}_{\mathsf{corr}}}$ are computed exactly as in Hybrid 0 (i.e. interacting with $\mathsf{CVZK.Sim}_1$), while $\{a_i\}_{i\in[\log\lambda]}$ are now computed using $\Sigma.\mathsf{Prv}_1(x, w)$ instead. Similarly, after the execution of the updated $\mathsf{CVZK.Sim}'_2$ the output $Z := (E, \{z_i\}_{i\in[\log\lambda]}, \{z^*_{\ell,j}\}^{j\in[\log\lambda]}_{\ell\in[n]})$ will be identically distributed to Hybrid 0 except $\{z_i\}_{i\in[\log\lambda]}$ that are now computed by $\Sigma.\mathsf{Prv}_2$ instead. Indistinguishability follows because the sHVZK property of Σ-protocol $\Sigma.\Pi$ is preserved under $\log\lambda$ parallel executions.

– *Indistinguishability between Hybrid 2 and Hybrid 1*: Next, we will argue that Hybrid 2 is indistinguishable from Hybrid 1 because of $\mathsf{InD}.\Pi$ is sHVZK against PPT distinguishers with access to $\mathcal{H}_S$. Namely, the output of $\mathsf{CVZK.Sim}'_1$ being $A := (\{a_i\}_{i\in[\log\lambda]}, \{a^*_{\ell,j}\}^{j\in[\log\lambda]}_{\ell\in[n]})$, $\langle c_\ell\rangle_{\ell\in[n]\setminus\mathcal{I}_{\mathsf{corr}}}$, where $\{a_i\}_{i\in[\log\lambda]}$ are identically distributed and $\{a^*_{\ell,j}\}^{j\in[\log\lambda]}_{\ell\in[n]}$ are parts of the input-delayed proof. In Hybrid 2, $\{a^*_{\ell,j}\}^{j\in[\log\lambda]}_{\ell\in[n]}$ are generated by using $\mathsf{InD.Sim}_1$; in Hybrid 1, for all

$\ell \in \mathcal{I}_{\mathsf{corr}}$, the corresponding $\{a^*_{\ell,j}\}^{j\in[\log\lambda]}_{\ell\in[n]}$ are computed in the exact same way (thus perfectly indistinguishable), while for $\ell \notin \mathcal{I}_{\mathsf{corr}}$, $\{a^*_{\ell,j}\}^{j\in[\log\lambda]}_{\ell\in[n]}$ are computed by $\mathsf{InD.Prv}_1$. Note that $\mathsf{InD.Prv}_1$ does not need to receive any extra input compared to $\mathsf{InD.Sim}_1$.

We now move to argue that the view of $\mathcal{A}$ is indistinguishable when running $\mathsf{CVZK.Sim}'_2$ (Hybrid 1) and $\mathsf{CVZK.Prv}_2$ (Hybrid 2). Let $Z := (E, \{z_i\}_{i\in[\log\lambda]}, \{z^*_{\ell,j}\}^{j\in[\log\lambda]}_{\ell\in[n]})$ be the output of $\mathsf{CVZK.Sim}'_2$. Observe that E is chosen uniformly at random, i.e. identically to real experiment where $\mathsf{CVZK.Prv}_2$ outputs the XOR of a random string R chosen by $\mathsf{CVZK.Prv}_1$ with the adversarial challenge. In addition, $\{z_i\}_{i\in[\log\lambda]}$ are computed the exact same way. Thus, it remains to argue about the indistinguishability of $\{z^*_{\ell,j}\}^{j\in[\log\lambda]}_{\ell\in[n]}$. First, note that for all honest verifiers, i.e. $\ell \notin \mathcal{I}_{\mathsf{corr}}$, $\mathsf{CVZK.Sim}'_2$ is identical to the real prover thus the produced z^*_ℓ's are perfectly indistinguishable. Let us finally discuss the corrupted verifiers, i.e. $\ell \in \mathcal{I}_{\mathsf{corr}}$. In Hybrid 1, we use $\mathsf{InD.Prv}_2$ to produce the corresponding $\{z^*_{\ell,j}\}^{j\in[\log\lambda]}_{\ell\in[n]}$, while in the real game $\mathsf{InD.Sim}_2$ is used. In order for $\mathsf{InD.Prv}_2$ to be able to output a valid input-delayed Σ proof it needs the actual witness to the corresponding statements being proved. As mentioned above, this is possible due to the use of $\mathcal{H}_S$. Indistinguishability follows due to the sHVZK property of the Σ-protocol $\mathsf{InD}.\Pi$ for every statement $x^*_{\ell,j}$, $\ell \in [n]$, $j \in [\log\lambda]$.

By the transitive property of indistinguishability of Hybrid 0 → Hybrid 1 → Hybrid 2, we conclude that $\mathbf{Expt}^{\mathsf{CVZK}}_{(\mathsf{Ideal},t_3,\mathcal{A},\mathcal{I}_{\mathsf{corr}})}(1^\lambda, x)$ and $\mathbf{Expt}^{\mathsf{CVZK}}_{(\mathsf{Real},t_3,\mathcal{A},\mathcal{I}_{\mathsf{corr}})}(1^\lambda, x, w)$ are indistinguishable. Hence, our construction is crowd-verifiable ZK against PPT adversaries with oracle access to $\mathcal{H}_S$. □

D Comments on the strength of the VMPC Security Model

We elaborate on the discussion intitiated in Section 5.

The necessity of trusting at least one server for privacy. Suppose that all servers $S_1, \ldots, S_k$ are corrupted, yet the private input x_ℓ is not leaked to the simulator Sim. In the real world, corrupting all servers enables the real world adversary $\mathcal{A}$ to completely run the **Compute** protocol. Since the users do not engage simultaneously, the environment can schedule U_ℓ to participate first. The correctness of the VMPC scheme Π implies that by receiving the data from U_ℓ, $\mathcal{A}$ can run **Compute** as if U_ℓ was the only participant and learn the evaluation of f for input $(\mathsf{abstain}, \ldots, \mathsf{abstain}, x_\ell, \mathsf{abstain}, \ldots, \mathsf{abstain})$. Therefore, it can infer information for x_ℓ that, in general, Sim can not obtain, for most functions of interest, as it only receives the evaluation of f on the entire input vector.

The necessity of trusting the clients for privacy. Since U_ℓ has no other communication channel with $S_1, \ldots, S_k$ other than via C_ℓ, and also lacks access to a trusted setup that could be used to provide cryptographic key information, it cannot prevented that C_ℓ will infer x_ℓ. To see this, consider the view of C_ℓ for two different values of x_ℓ that result in different outputs with respect to

f; in such case, C_ℓ can mount the following attack to distinguish between the two possible values of x_ℓ: simulate $S_1, \ldots, S_k$, for fixed values for all users other than U_ℓ, and exploit the correctness of the VMPC evaluation to compute f and hence infer x_ℓ. Thus, if $C_\ell \in L_{\mathsf{corr}}$ but the private input x_ℓ is not leaked to the simulator Sim, then the ideal and the real world can be distinguished.

The necessity of trusting the verifier for end-to-end verifiability. Clearly, in the ideal world setting $\mathcal{F}^{f,R}_{\mathsf{vmpc}}(\mathcal{P})$, plays the role of a trusted verifier which output depends on the fixed relation R that sets the rules of successful verification. However, in the real world, if $\mathcal{A}$ corrupts the verifier, then it can recommend arbitrary verdicts regardless if the intended evaluation y and the value y' that $\mathcal{A}$ returns is in relation R or not. Recall that in our setting, users consult the verifier for end-to-end verifiability since they are computationally limited.

E Supplementary Material for Section 6

E.1 The intuition behind Theorem 3

According to the statement of Theorem 3, if R is not a δ-spreading relation over $\mathsf{Img}[f]$, then for given values of κ, δ, there is a lower bound $\min\{2^{-\kappa\delta} - \epsilon(\lambda), \gamma(\lambda)\}$ on the best possible error in the VMPC computation of f, if the latter is feasible. For simplicity, ignoring the negligible term $\epsilon(\lambda)$ and considering the non-negligible value $\gamma(\lambda)$ sufficiently large, in the rest of this discussion we can assume that the lower bound for the VMPC security error is practically $2^{-\kappa\delta}$, when $\kappa > 0$. Given this assumption, we make the following interesting observations.

Reaching the optimal expectation in VMPC feasibility. Although not δ-spreading, it can be the case (yet not necessarily) that R is δ'-spreading over $\mathsf{Img}[f]$, obviously for some value $\delta' < \delta$. Let $\mathsf{Spr}_R \subseteq \mathbb{R}_{\geq 0}$ be the set non-negative values δ' s.t. R is δ'-spreading over $\mathsf{Img}[f]$. Note that we can theoritically consider cases where $\mathsf{Spr}_R = \emptyset$, yet we choose to restrict to the interesting scenario where Spr_R contains at least one value. Hence, given that Spr_R is non-empty, there is a *supremum value* $\delta^* := \sup\{\mathsf{Spr}_R\}$ s.t. R is δ^*-spreading over $\mathsf{Img}[f]$. By the definition of δ^* and Theorem 3, the value $2^{-\kappa\delta^*}$ sets the best possible error (ignoring negligible cryptographic error terms) that one can expect for the VMPC feasibility of f w.r.t. R. As it is shown in Theorem 4, our VMPC construction meets this essentially optimal expectation for the special case where $\kappa = 1$.

The necessity for non-deterministic human behavior. The above observation reveals a crucial requirement on the "behavior" of the human users when they engage in the VMPC execution. Namely, if the users engage in a deterministic way, i.e. $\kappa = 0$, then VMPC feasibility with a reasonably small error is not possible *whatever* the value of δ^*[9]. The latter fact captures the intuition that against an all-malicious environment, the humans must behave in a way

[9] In particular, since $\delta^* \leq n < +\infty$, we have that $2^{-\kappa\delta^*} = 1$, so the best possible VMPC security error that we can expect is the non-negligible value $\gamma(\lambda)$.

that is somehow "unpredictable" in the view of the adversary. This is a conclusion that not only formally expresses, but also extends to the general MPC setting the intuition behind the design of all state-of-the-art end-to-end verifiable e-voting systems. There, the (human) voters are encouraged to engage in a "cast-or-audit" ballot verification mechanism which can be seen as a form of cut-and-choose proofs.

E.2 Proof of Theorem 3

Proof (sketch). Assume that condition (1) of Theorem 3 does not hold. Then, by Definition 7, there exist $\mathbf{x} = (x_1, \ldots, x_n), \mathbf{x}' = (x'_1, \ldots, x'_n) \in X^n$ s.t.

$$\mathsf{Dcr}_n(\mathbf{x}, \mathbf{x}') \le \delta \text{ AND } \neg R\big(f(\mathbf{x}), f(\mathbf{x}')\big) \ .$$

Let $\Pi^{\mathcal{G}_{\mathsf{BB}},\mathcal{F}_{\mathsf{sc}}}$ be a VMPC scheme operating with the parties in $\mathcal{P}$ with user min entropy κ that achieves correctness and a helper functionality $\mathcal{H}$. We will show that condition (2) holds. Namely, there is a negligible function ϵ and a non-negligible function γ such that $\Pi^{\mathcal{G}_{\mathsf{BB}},\mathcal{F}_{\mathsf{sc}}}$ can not $\mathcal{H}$-EUC realize $\mathcal{F}^{f,R}_{\mathsf{vmpc}}(\mathcal{P})$ with error less than $\min\{2^{-\kappa\delta} - \epsilon(\lambda), 2^{-\kappa}\gamma(\lambda)\}$.

We denote by y, y' the evaluations $f(\mathbf{x}), f(\mathbf{x}')$. respectively. We also denote by τ, τ' the transcript committed on the BB when execution is run on inputs $\mathbf{x}, \mathbf{x}'$, respectively

Since $\mathsf{Dcr}_n X^n(\mathbf{x}, \mathbf{x}') \le \delta$, there is a subset of indices $\mathcal{I}_{\mathsf{diff}} \in [n]$ of size δ that contains all positions where components of the vectors $\mathbf{x}, \mathbf{x}'$ differ. Formally, we have that

$$|\mathcal{I}_{\mathsf{diff}}| \le \delta \text{ and } \forall \ell \in [n] \setminus \mathcal{I}_{\mathsf{diff}} : x_\ell = x'_\ell$$

Let $\alpha_\ell, \alpha'_\ell$ be the individual audit data of user U_ℓ when engaging on input x_ℓ. We distinguish the following cases:

Case 1: *The random variables $\langle(\alpha_1, \ldots, \alpha_n), \tau'\rangle$ and $\langle(\alpha'_1, \ldots, \alpha'_n), \tau'\rangle$ are indistinguishable.*

This case refers to the concept of delegatable verification. Namely, the individual audit data should not leak information about the user's input to the verifier. This feature is desirable for privacy, but it can be exploited by an adversary that manages to guess the users' coins to break end-to-end verifiability as described below.

We construct a real world adversary $\mathcal{A}_{\mathsf{e2e}}$ against the end-to-end verifiability of $\Pi^{\mathcal{G}_{\mathsf{BB}},\mathcal{F}_{\mathsf{sc}}}$, that executes the following steps:

1. It manages the VMPC execution by corrupting all the servers and the users' clients.
2. For every $\ell \in \mathcal{I}_{\mathsf{diff}}$, it guesses the randomness, r_ℓ, of user U_ℓ as the string $\tilde{r}_\ell$.
3. It runs the VMPC execution normally except the following modification: for every $\ell \in \mathcal{I}_{\mathsf{diff}}$, regardless of the input of U_ℓ, it interacts with U_ℓ as if the latter was provided with input x'_ℓ and generated internal randomness $\tilde{r}_\ell$.
4. It runs the **Compute** protocol honestly on input $\mathbf{x}'$ by running all servers.

5. It allows the verifier V to interact normally and send the message (y', v) to each U_ℓ, $\ell \in [n]$.

Let G_{e2e} be the event that for every $\ell \in \mathcal{I}_{\mathsf{diff}}$, $\mathcal{A}_{\mathsf{e2e}}$ guesses the coins of the users U_ℓ, $\ell \in \mathcal{I}_{\mathsf{diff}}$. By independency of the users' interaction and the fact that (a) $|\mathcal{I}_{\mathsf{diff}}| \leq \delta$ and (b) the user min entropy in Π is κ, we have that

$$\Pr[G_{\mathsf{e2e}}] = \prod_{\ell \in \mathcal{I}_{\mathsf{diff}}} \Pr\left[\mathcal{A}_{\mathsf{e2e}} \text{ guesses } \tilde{r}_\ell = r_\ell\right] \geq 2^{-\kappa\delta} \,. \tag{23}$$

Assume that G_{e2e} occurs. Then, by corrupting all servers and clients, $\mathcal{A}_{\mathsf{e2e}}$ can totally control the users' interaction and the individual audit data that they provide to the verifier V. In particular, V perform verification given (i) a collection of individual audit data $(\alpha_1, \ldots, \alpha_n)$ corresponding to $\mathbf{x}$ and (ii) a BB transcript τ' that corresponds to $\mathbf{x}'$.

Observe that in an honest VMPC execution, where the individual audit data $(\alpha'_1, \ldots, \alpha'_n)$ corresponding to $\mathbf{x}'$, V will send the value 1 to all users if the execution was run honestly. Besides, by the assumption for Case 1, $\langle(\alpha_1, \ldots, \alpha_n), \tau'\rangle$ and $\langle(\alpha'_1, \ldots, \alpha'_n), \tau'\rangle$ are indistinguishable by any PPT algorithm, hence also by V. Therefore, according to description of $\mathcal{A}_{\mathsf{e2e}}$, if for every $\ell \in \mathcal{I}_{\mathsf{diff}}$: (i) U_ℓ has input x_ℓ and (ii) $\mathcal{A}_{\mathsf{e2e}}$ guesses the randomness of U_ℓ (i.e. $\tilde{r}_\ell = r_\ell$), then the PPT verifier V can not distinguish between α'_ℓ and the honest execution data α_ℓ, where it would be accepting verification. The latter implies that on input $\langle(\alpha_1, \ldots, \alpha_n), \tau'\rangle$, the verifier V will send $(y', v_\ell = 1)$ to each U_ℓ, $\ell \in [n]$ with probability at least $1 - \mathsf{negl}(\lambda)$.where $\langle\epsilon_\ell(\cdot)\rangle_{\ell \in \mathcal{I}_{\mathsf{diff}}}$ are negligible functions.Given the above, consider the following environment that takes advantage of $\mathcal{A}_{\mathsf{e2e}}$'s strategy :

$-\mathcal{Z}_{y',\mathsf{Ver}}$: selects $\mathbf{x} \in X^n$ and sends the message $(\textsc{Cast}, \mathsf{sid}, x_\ell)$ as input to each user U_ℓ, $\ell \in [n]$. It instructs the full corruption of the servers and the clients. It outputs 1 iff all users return the message $(\textsc{Result}, \mathsf{sid}, y', 1)$.

By the above analysis and Eq. (23), in the real world setting, we have that for any helper $\mathcal{H}$

$$\begin{aligned} &\Pr\left[\mathsf{EXEC}^{\mathcal{P},\Pi^{\mathcal{G}_{\mathsf{BB}},\mathcal{F}_{\mathsf{sc}}}}_{\mathcal{A}_{\mathsf{e2e}},\mathcal{Z}_{y',\mathsf{Ver}},\mathcal{H}}(\lambda) = 1\right] \geq \\ &\geq \Pr[G_{\mathsf{e2e}}] \cdot \Pr\left[\mathsf{EXEC}^{\mathcal{P},\Pi^{\mathcal{G}_{\mathsf{BB}},\mathcal{F}_{\mathsf{sc}}}}_{\mathcal{A}_{\mathsf{e2e}},\mathcal{Z}_{y',\mathsf{Ver}},\mathcal{H}}(\lambda) = 1 \;\middle|\; G_{\mathsf{e2e}}\right] \geq \\ &\geq 2^{-\kappa\delta} \cdot \Big(1 - \mathsf{negl}(\lambda)\Big) = 2^{-\kappa\delta} - \mathsf{negl}(\lambda). \end{aligned} \tag{24}$$

On the other hand, in the ideal world, if the environment's input vector is $\mathbf{x}$, then any ideal simulator Sim with access to $\mathcal{H}$ can follow either two of the possible strategies:

(S.1). Neglect the strategy of $\mathcal{A}_{\mathsf{e2e}}$ and send the message $(\textsc{Verify\_response}, \mathsf{sid}, U_\ell, y'', \tilde{v})$ to $\mathcal{F}^{f,R}_{\mathsf{vmpc}}(\mathcal{P})$, for some $\ell \in [n]$, where $y'' \neq y'$ and $\tilde{v} \in \{0, 1\}$.

(S.2). Simulate an execution respecting $\mathcal{A}_{\mathsf{e2e}}$'s steps and send $(\textsc{Verify\_response}, \mathsf{sid}, U_\ell, y', \tilde{v})$ to $\mathcal{F}^{f,R}_{\mathsf{vmpc}}(\mathcal{P})$ for all $\ell \in [n]$.

Whenever the simulator follows (S.1), the environment $\mathcal{Z}_{y',\mathsf{Ver}}$ will never output 1. Indeed, note that in any case, $\mathcal{F}_{\mathsf{vmpc}}^{f,R}(\mathcal{P})$ will send either $(\textsc{Result}, \mathsf{sid}, y'', 1)$ or $(\textsc{Result}, \mathsf{sid}, y'', 0)$ to U_ℓ, where $y'' \neq y'$. Moreover, whenever the simulator follows (S.2), $\mathcal{Z}_{y',\mathsf{Ver}}$ will never output 1 as by assumption $\neg R(y, y')$ so $\mathcal{F}_{\mathsf{vmpc}}^{f,R}(\mathcal{P})$ will send $(\textsc{Result}, \mathsf{sid}, y', 0)$ to all users. We conclude that

$$\Pr\left[\mathsf{EXEC}_{\mathsf{Sim},\mathcal{Z}_{y',\mathsf{Ver}},\mathcal{H}}^{\mathcal{F}_{\mathsf{vmpc}}^{f,R}(\mathcal{P}),\mathcal{G}_{\mathsf{BB}}}(\lambda) = 1\right] = 0\ .$$

Therefore, by Eq. (24), therefore there exists a negligible function $\epsilon(\lambda)$ s.t. $\mathcal{Z}_{y',\mathsf{Ver}}$ has distinguishing advantage at least

$$\begin{aligned}\Big|\Pr&\big[\mathsf{EXEC}_{\mathsf{Sim},\mathcal{Z}_{y',\mathsf{Ver}},\mathcal{H}}^{\mathcal{F}_{\mathsf{vmpc}}^{f,R}(\mathcal{P}),\mathcal{G}_{\mathsf{BB}}}(\lambda) = 1\big]-\\ &-\Pr\left[\mathsf{EXEC}_{\mathcal{A}_{\mathsf{e2e}},\mathcal{Z}_{y',\mathsf{Ver}},\mathcal{H}}^{\mathcal{P},\Pi^{\mathcal{G}_{\mathsf{BB}},\mathcal{F}_{\mathsf{sc}}}}(\lambda) = 1\right] \geq 2^{-\kappa\delta} - \epsilon(\lambda)\ .\end{aligned}$$

Case 2: *There exists a PPT algorithm $\mathcal{D}^*$ that distinguishes $\langle(\alpha_1, \ldots, \alpha_n), \tau'\rangle$ and $\langle(\alpha'_1, \ldots, \alpha'_n), \tau'\rangle$ with non-negligible advantage $\beta(\lambda)$.*

Since all users derive their individual audit data independently and α_ℓ is identically distributed to α'_ℓ for $\ell \notin \mathcal{I}_{\mathsf{diff}}$, we can assume that $\mathcal{D}^*$ distinguishes $(\alpha_\ell)_{\ell\in\mathcal{I}_{\mathsf{diff}}}$ and $(\alpha'_\ell)_{\ell\in\mathcal{I}_{\mathsf{diff}}}$ with non-negligible advantage $\beta(\lambda)$. W.l.o.g, assume that

$$\Pr\left[\mathcal{D}^*\big((\alpha_\ell)_{\ell\in\mathcal{I}_{\mathsf{diff}}}\big) = 1\right] - \Pr\left[\mathcal{D}^*\big((\alpha'_\ell)_{\ell\in\mathcal{I}_{\mathsf{diff}}}\big) = 0\right] \geq \beta(\lambda)\ .$$

For $|\mathcal{I}_{\mathsf{diff}}| \leq \delta \leq \frac{n}{2}$, we construct the environment $\mathcal{Z}^*$ that operates as follows:

1. It selects randomly a bit $b \in \{0, 1\}$.
2. If $b = 0$, it sets the users' input vector $\tilde{\mathbf{x}} = (\tilde{x}_1, \ldots, \tilde{x}_n)$ as

$$\tilde{x}_\ell := \begin{cases} x_\ell, & \text{if } 1 \leq \ell \leq |\mathcal{I}_{\mathsf{diff}}| \\ x'_\ell, & \text{if } |\mathcal{I}_{\mathsf{diff}}| + 1 \leq \ell \leq 2|\mathcal{I}_{\mathsf{diff}}| \\ \mathsf{abstain}, & \text{otherwise} \end{cases}$$

3. If $b = 1$, it sets the users' input vector $\tilde{\mathbf{x}} = (\tilde{x}_1, \ldots, \tilde{x}_n)$ as above by swapping the role of x_ℓ and x'_ℓ.
4. It instructs the corruption of V and schedules the rest of the execution normally.
5. It obtains the audit data $(\tilde{\alpha}_1, \ldots, \tilde{\alpha}_{|\mathcal{I}_{\mathsf{diff}}|})$ from V and provides them as input to $\mathcal{D}^*$.
6. If $\mathcal{D}^*$ guesses b, then it returns 1. Otherwise, it returns a random bit.

Assume now a real world execution scheduled by $\mathcal{Z}^*$ which is dummy except from the fact that it corrupts V in order to forward the individual audit data to $\mathcal{Z}^*$. Then, clearly, it holds that

$$
\begin{aligned}
&\Pr\left[\mathsf{EXEC}^{\mathcal{P},\Pi^{\mathcal{G}_{\mathsf{BB}},\mathcal{F}_{\mathsf{sc}}}}_{\mathcal{A}^*,\mathcal{Z}^*,\mathcal{H}}(\lambda)=1\right]=\\
=&\frac{1}{2}\cdot\Pr\left[\mathsf{EXEC}^{\mathcal{P},\Pi^{\mathcal{G}_{\mathsf{BB}},\mathcal{F}_{\mathsf{sc}}}}_{\mathcal{A}^*,\mathcal{Z}^*,\mathcal{H}}(\lambda)=1\;\middle|\;b=1\right]+\\
&+\frac{1}{2}\cdot\Pr\left[\mathsf{EXEC}^{\mathcal{P},\Pi^{\mathcal{G}_{\mathsf{BB}},\mathcal{F}_{\mathsf{sc}}}}_{\mathcal{A}^*,\mathcal{Z}^*,\mathcal{H}}(\lambda)=1\;\middle|\;b=0\right]=\\
=&\frac{1}{2}\cdot\Big(\Pr\left[\mathcal{D}^*\left(\alpha_\ell\right)_{\ell\in\mathcal{I}_{\mathsf{diff}}}\right)=1\;\middle|\;b=1\right]\cdot 1\\
&+\Pr\left[\mathcal{D}^*\left(\alpha_\ell\right)_{\ell\in\mathcal{I}_{\mathsf{diff}}}\right)=0\;\middle|\;b=1\right]\cdot\frac{1}{2}\Big)+\\
&+\frac{1}{2}\cdot\Big(\Pr\left[\mathcal{D}^*\left(\alpha'_\ell\right)_{\ell\in\mathcal{I}_{\mathsf{diff}}}\right)=0\;\middle|\;b=0\right]\cdot 1\\
&+\Pr\left[\mathcal{D}^*\left(\alpha'_\ell\right)_{\ell\in\mathcal{I}_{\mathsf{diff}}}\right)=1\;\middle|\;b=0\right]\cdot\frac{1}{2}\Big)+\\
=&\frac{1}{2}+\frac{1}{2}\Big(\Pr\left[\mathcal{D}^*\left(\alpha_\ell\right)_{\ell\in\mathcal{I}_{\mathsf{diff}}}\right)=1\right]-\Pr\left[\mathcal{D}^*\left(\alpha'_\ell\right)_{\ell\in\mathcal{I}_{\mathsf{diff}}}\right)=0\right]\Big)\geq\\
\geq&\frac{1}{2}+\frac{\beta(\lambda)}{2}\;.
\end{aligned}
\tag{25}
$$

On the other hand, in the ideal world, an arbitrary simulator Sim with oracle access to $\mathcal{H}$ running under the instructions of $\mathcal{Z}^*$, will send the message $(\text{VERIFY\_RESPONSE},\mathsf{sid},U_{\ell^*},y^*,\tilde{v})$ to $\mathcal{F}^{f,R}_{\mathsf{vmpc}}(\mathcal{P})$ without obtaining x_{ℓ^*}, since the users' clients are not corrupted. By construction of $\mathcal{Z}^*$, $\mathcal{F}^{f,R}_{\mathsf{vmpc}}(\mathcal{P})$ will always output the value $\tilde{y}=f(\tilde{\mathbf{x}})$ for any value of $b\in\{0,1\}$, given that f is a symmetric function. Therefore, Sim's steps will be independent of b and it holds that

$$\Pr\left[\mathsf{EXEC}^{\mathcal{F}^{f,R}_{\mathsf{vmpc}}(\mathcal{P}),\mathcal{G}_{\mathsf{BB}}}_{\mathsf{Sim},\mathcal{Z}^*,\mathcal{H}}(\lambda)=1\right]=\frac{1}{2} \tag{26}$$

By Eq. (25) and (26), we have that the distinguishing advantage of $\mathcal{Z}^*$ between the ideal-world and the real-world execution is $\frac{\beta(\lambda)}{2}$, which is non-negligible.

Completing the proof. Combining the results from Cases 1 and 2 and setting $\gamma(\lambda)=\frac{\beta(\lambda)}{2}$, we get that for any VMPC scheme Π and any helper $\mathcal{H}$, it holds that there is a real-world adversary ($\mathcal{A}_{\mathsf{e2e}}$ or $\mathcal{A}^*$) and an environment ($\mathcal{Z}_{y'_{\mathsf{Ver}}}$ or $\mathcal{Z}^*$) such that for every simulator Sim, it holds that

$$
\begin{aligned}
\Big|\Pr&\left[\mathsf{EXEC}^{\mathcal{F}^{f,R}_{\mathsf{vmpc}}(\mathcal{P}),\mathcal{G}_{\mathsf{BB}}}_{\mathsf{Sim},\mathcal{Z},\mathcal{H}}(\lambda)=1\right]-\\
&-\Pr\left[\mathsf{EXEC}^{\mathcal{P},\Pi^{\mathcal{G}_{\mathsf{BB}},\mathcal{F}_{\mathsf{sc}}}}_{\mathcal{A}^*,\mathcal{Z}^*,\mathcal{H}}(\lambda)=1\right]\Big|\geq\min\{2^{-\kappa\delta}-\epsilon(\lambda),\gamma(\lambda)\}.
\end{aligned}
$$

Namely, Π does not $\mathcal{H}$-realize $\mathcal{F}^{f,R}_{\mathsf{vmpc}}(\mathcal{P})$ with error less than $\min\{2^{-\kappa\delta}-\epsilon(\lambda),\gamma(\lambda)\}$. □

F Supplementary Material for VMPC Construction

F.1 Dual-mode Homomorphic Commitments

To enable VMPC, similar as [4], the MPC servers need to commit to every shared value to the bulletin board. The commitment scheme should be additively homomorphic (for both message and randomness); however, commitment schemes like lifted ElGamal (presented above) or Pedersen commitment are not sufficient in our setting. Given that in VMPC the commitment key is generated by the servers, Pedersen commitment becomes not binding when all the servers are malicious, while ElGamal commitment does not allow the simulator to equivocate. To overcome these problems, we make use of dual-mode homomorphic commitments which allow for two ways to choose the commitment key such that the commitment is either perfectly binding or equivocal. In the following, we adopt and modify the dual-mode commitment definition proposed by [52] to against helger-aided PPT adversaries.

Definition 12. *A dual-mode commitment scheme is a tuple of PPT algorithms* $\mathsf{DC.Gen}, \mathsf{DC.Com}, \mathsf{DC.Sim}$ *such that*

1. $\mathsf{DC.Gen}(1^\lambda)$ *outputs a commitment key denoted* ck*.*
2. *When* $\mathsf{ck} \leftarrow \mathsf{DC.Gen}(1^\lambda)$ *and* $m \in \{0,1\}^\lambda$ *the algorithm* $\mathsf{DC.Com}_{\mathsf{ck}}(m;r)$ *with a random* r *is a non-interactive perfectly-binding commitment scheme with de-commitment algorithm* $\mathsf{DC.Open}$ *and de-commitment verification algorithm* $\mathsf{DC.Ver}$*. We require that*

   $\mathsf{DC.Ver}_{\mathsf{ck}}(\mathsf{DC.Com}_{\mathsf{ck}}(m;r), \mathsf{DC.Open}_{\mathsf{ck}}(m;r)) = m$ *except with negligible probability.*
3. *For every PPT adversary* $\mathcal{A}$ *aided by* $\mathcal{H}_S$ *and every polynomial* $p(\cdot)$*, the output of the following two experiments is computationally indistinguishable:*

   $\underline{\mathbf{Expt}^{Real}_{\mathsf{Com},\mathcal{A}}(1^\lambda)}$

   - $\mathsf{ck} \leftarrow \mathsf{DC.Gen}(1^\lambda)$
   - *For* $i = 1, \ldots, p(\lambda)$*:*
     1. $m_i \leftarrow \mathcal{A}(\mathsf{ck}, \mathbf{c}, \mathbf{r})$
     2. $r_i \leftarrow \{0,1\}^{poly(1^\lambda)}$
     3. $c_i = \mathsf{DC.Com}_{\mathsf{ck}}(m_i; r_i)$
     4. *Set* $\mathbf{c} = c_1, \ldots, c_i$ *and* $\mathbf{r} = r_1, \ldots, r_i$
   - *Output* $\mathcal{A}(\mathsf{ck}_i, m_1, r_1, \ldots, m_{p(\lambda)}, r_{p(\lambda)})$

```
$\underline{\mathbf{Expt}^{Simulation}_{\mathsf{DC.Sim}}(1^\lambda)}$
- $\mathsf{ck} \leftarrow \mathsf{DC.Gen}(1^\lambda)$
- For $i = 1, \ldots, p(\lambda)$:
    1. $c_i = \mathsf{DC.Sim}$
    2. $m_i \leftarrow \mathcal{A}(\mathsf{ck}, \mathbf{c}, \mathbf{r})$
    3. $r_i \leftarrow \mathsf{DC.Sim}(m_i)$
    4. Set $\mathbf{c} = c_1, \ldots, c_i$ and $\mathbf{r} =$
$r_1, \ldots, r_i$
- Output
  $\mathcal{A}(\mathsf{ck}_i, m_1, r_1, \ldots, m_{p(\lambda)}, r_{p(\lambda)})$
```

Our construction is built on top of the perfectly binding commitment as proposed in C.2. The resulting commitment should be either perfectly binding or perfectly equivocatable, depending on the commitment key. Consistently with our setting, we require that the servers jointly generate the commitment key and post it in the $\mathcal{G}_{\mathsf{BB}}$. Then, commitment executions may run among servers or between servers and clients. We denote by $\mathsf{CS}^{\mathbf{F}} = (\mathsf{CS}^{\mathbf{F}}.\mathsf{Gen}, \mathsf{CS}^{\mathbf{F}}.\mathsf{Com}, \mathsf{CS}^{\mathbf{F}}.\mathsf{Ver})$ the perfectly binding commitment that is secure against helper-aided PPT adversaries. Our dual mode commitment scheme DC works as follows:

$\mathsf{DC.Gen}(1^\lambda)$:

- Generate $\mathsf{ck} \leftarrow \mathsf{CS}^{\mathbf{F}}.\mathsf{Gen}(1^\lambda)$;
- Commit $\mathbf{c} \leftarrow \mathsf{CS}^{\mathbf{F}}.\mathsf{Com}(1; \mathbf{t})$ with fresh randomness $\mathbf{t} \leftarrow (\mathbb{Z}_q)^k$;
- Output the commitment key as $\mathsf{ck}^* := (\mathsf{ck}, \mathbf{c})$.

$\mathsf{DC.Com}_{\mathsf{ck}^*}(m; \mathbf{r})$:

- To commit a message $m \in \mathbb{Z}_q$, return commitment $\mathbf{c}^* \leftarrow \mathbf{c}^m \cdot \mathsf{CS}^{\mathbf{F}}.\mathsf{Com}(0; \mathbf{r})$.

$\mathsf{DC.Open}_{\mathsf{ck}}(c, m, \mathbf{r})$:

- To open commitment c, return $(m, \mathbf{r})$.

$\mathsf{DC.Ver}_{\mathsf{ck}}(c, m, \mathbf{r})$:

- Output 1 iff $\mathbf{c}^* = \mathbf{c}^m \cdot \mathsf{CS}^{\mathbf{F}}.\mathsf{Com}(0; \mathbf{r})$.

By notation $\mathbf{c}^m$, we meant element-wisely power each element of $\mathbf{c}$ of m. Similarly, $\mathbf{c}_1 \cdot \mathbf{c}_2$ stands for element-wise product (a.k.a. Hadamard product).

Clearly, the commitment scheme is perfect binding when $\mathbf{c}$ is a commitment of a non-zero message, say 1 w.r.t. $\mathsf{CS}^{\mathbf{F}}$ as proposed in C.2. However, if we set the commitment key with $\mathbf{c} \leftarrow \mathsf{CS}^{\mathbf{F}}.\mathsf{Com}(0; \mathbf{t})$, then it is perfectly equivocatable, given the trapdoor information $\mathsf{td} := \mathbf{t}$. More specifically, we can open $\mathsf{DC.Com}_{\mathsf{ck}}(m; \mathbf{r})$ to any given $m' \in \mathbb{Z}_q$ and $\mathbf{r}' = \mathbf{t}(m - m') + \mathbf{r}$. During our proof, the simulator will switch these two types of commitment keys via rewinding.

F.2 A Σ-Protocol for Beaver Triples

We now propose a Σ-protocol for proving A, B, C commits to a, b, c and $ab = c$. The protocol is depicted in Fig. 8.

Common input: ck.
Statement: $A := \mathsf{DC.Com}_{\mathsf{ck}}(a; \mathbf{r_1})$, $B := \mathsf{DC.Com}_{\mathsf{ck}}(b; \mathbf{r_2})$, and $C := B^a \cdot \mathsf{DC.Com}_{\mathsf{ck}}(0; \mathbf{r_3})$.
Witness: $a \in \mathbb{Z}_p$, $\mathbf{r_1}, \mathbf{r_3} \in (\mathbb{Z}_p)^k$.

- $\Sigma.\mathsf{Prv}_1\big(x := (A, B, C), w := (a, \mathbf{r_1}, \mathbf{r_3})\big)$:
    - Pick $t \leftarrow \mathbb{Z}_q$ and $\mathbf{s}, \mathbf{y} \leftarrow (\mathbb{Z}_q)^k$;
    - Compute $T = B^t \cdot \mathsf{DC.Com}_{\mathsf{ck}}(0; \mathbf{s})$, $Y = \mathsf{DC.Com}_{\mathsf{ck}}(t; \mathbf{y})$;
    - Send T, Y to the verifier and set the state $\mathsf{st} := (w, t, \mathbf{s}, \mathbf{y})$.
- $\Sigma.\mathsf{Prv}_2(\mathsf{st}, e)$: after receiving the challenge $e \in \{0, 1\}^\lambda$ from the verifier, compute $t' = a \cdot e + t$, $\mathbf{s}' = \mathbf{r_3} \cdot e + \mathbf{s}$ and $\mathbf{y}' = \mathbf{r_1} \cdot e + \mathbf{y}$; send the response $(t', \mathbf{s}', \mathbf{y}')$ to the verifier.
- $\Sigma.\mathsf{Verify}\big((A, B, C), (T, Y), e, (t', \mathbf{s}', \mathbf{y}')\big)$: accept the proof if and only if:

$$C^e \cdot T = B^{t'} \cdot \mathsf{DC.Com}_{\mathsf{ck}}(0; \mathbf{s}') \text{ and } A^e \cdot Y = \mathsf{DC.Com}_{\mathsf{ck}}(t'; \mathbf{y}') \ .$$

Fig. 8: A Σ-protocol for commitments of Beaver triples.

Theorem 7. *The protocol presented in Fig. 8 is a Σ-protocol for knowledge of $a, \mathbf{r_1}, \mathbf{r_3}$ such that $A := \mathsf{Com}_{\mathsf{ck}}(a; \mathbf{r_1})$ and $C := B^a \cdot \mathsf{Com}_{\mathsf{ck}}(0; \mathbf{r_3})$.*

Proof. For perfect completeness, it is easy to verify that all the verification equations hold when the prover is honest. For special soundness, assume we have two valid transcript $(T, Y, e_1, t'_1, \mathbf{s'_1}, \mathbf{y'_1})$ and $(T, Y, e_2, t'_2, \mathbf{s'_2}, \mathbf{y'_2})$ for $e_1 \neq e_2$. The knowledge extractor computes $a = \frac{t'_1 - t'_2}{e_1 - e_2}$, $\mathbf{r_1} = \frac{\mathbf{y'_1} - \mathbf{y'_2}}{e_1 - e_2}$, and $\mathbf{r_3} = \frac{\mathbf{s'_1} - \mathbf{s'_2}}{e_1 - e_2}$. For special honest verifier zero-knowledge, we have to construct a simulator that given any challenge e can output a simulated transcript that is indistinguishable to the real one. On challenge e, the simulator randomly picks $t' \leftarrow \mathbb{Z}_p$ and $\mathbf{s}', \mathbf{y}' \leftarrow (\mathbb{Z}_p)^k$ and computes $T = B^{t'} \cdot \mathsf{Com}_{\mathsf{ck}}(0; \mathbf{s}') \cdot C^{-e}$ and $Y = \mathsf{Com}_{\mathsf{ck}}(t'; \mathbf{y}') \cdot A^{-e}$. It is straightforward that the simulated $(T, Y, e, t', \mathbf{s}', \mathbf{y}')$ has identical distribution to the real one. □

F.3 EUC helper $\mathcal{H}$

The EUC helper definition is slightly different from the CVZK helper. Namely, it is an externalized functionality that also interacts with the environment $\mathcal{Z}$, which allows $\mathcal{Z}$ to inform the EUC helper about which parties are corrupted. Otherwise, it is identical to the CVZK helper.

The helper $\mathcal{H}$ functionality

It is parameterized by an adaptive one-way function f and a set S.
Initially, $S = \emptyset$.

– Upon receiving $(\mathsf{sid}, \mathsf{Corrupt}, \mathsf{tag}^*)$ from the environment $\mathcal{Z}$, set $S = S \cup \{\, \mathsf{tag}^* \,\}$.

– Upon receiving $(\mathsf{sid}, \mathsf{Query}, \mathsf{tag}, \beta)$, if $\mathsf{tag} \in S$, then it returns a value $\alpha \in X_{\mathsf{tag}}$ s.t. $f_{\mathsf{tag}}(\alpha) = \beta$. Otherwise, it returns $\perp$.

Fig. 9: The helper functionality $\mathcal{H}$.

F.4 Straight-line simulatable zero-knowledge

To prove security of the construction, the VMPC simulator must be able to simulate several ZK proofs, such as proving the correctness of shared Beaver triples. However, in the EUC security setting, the simulator cannot rewind the adversarial verifier, and thus the underlying ZK proofs must be straight-line simulatable. In Fig. 10, we describe a three-move ZK protocol that is straight-line simulatable (SLZK), when the simulator has access to the helper family described in Fig. 9, that is essentially a simplification of our CVZK compiler, adjusted as a standard ZK proof between a prover and a single (high entropy) verifier, without the use of a coalescence function. Our SLZK protocol is a compiler for any Σ-protocol $\Sigma.\Pi = (\Sigma.\mathsf{Prv}_1, \Sigma.\mathsf{Prv}_2, \Sigma.\mathsf{Verify})$ for $\mathbf{NP}$ language $\mathcal{L}$ and $(x, w) \in \mathcal{R}_{\mathcal{L}}$, and makes use of an input-delayed $\mathsf{InD}.\Sigma$-protocol $\mathsf{InD}.\Sigma.\Pi = (\mathsf{InD}.\Sigma.\mathsf{Prv}_1, \mathsf{InD}.\Sigma.\mathsf{Prv}_2, \mathsf{InD}.\Sigma.\mathsf{Verify})$ (for the family of languages $\left\{\mathcal{L}^*_{\mathsf{tag}_\ell}\right\}_{\ell \in [n]}$ as defined in Section 4.3) and a publicly samplable adaptive one-way function family $\mathbf{F} = \{f_{\mathsf{tag}} : X_{\mathsf{tag}} \mapsto Y_{\mathsf{tag}}\}_{\mathsf{tag} \in \{0,1\}^\lambda}$. We make use of fixed member f_{tag} of the family $\mathbf{F}$ and the same helper $\mathcal{H}_S$ where S contains all the tags of the malicious parties.

Theorem 8. *Let* $\mathbf{F} = \{f_{\mathsf{tag}} : X_{\mathsf{tag}} \mapsto Y_{\mathsf{tag}}\}_{\mathsf{tag} \in \{0,1\}^{\lambda / \log^2 \lambda}}$ *be a publicly samplable adaptive one-way function family. For every Sigma protocol* $\Sigma.\Pi = (\Sigma.\mathsf{Prv}_1, \Sigma.\mathsf{Prv}_2, \Sigma.\mathsf{Verify})$ *for some language* $\mathcal{L} \in \mathbf{NP}$*, every input-delayed Σ-protocol* $\mathsf{InD}.\Sigma.\Pi = (\mathsf{InD}.\Sigma.\mathsf{Prv}_1, \mathsf{InD}.\Sigma.\mathsf{Prv}_2, \mathsf{InD}.\Sigma.\mathsf{Verify})$ *for language* $\left\{\mathcal{L}^*_{\mathsf{tag}_\ell}\right\}_{\ell \in [n]}$ *and every* $\mathsf{tag} \in \{0,1\}^\lambda$*, the protocol presented in Fig. 10 is a ZK protocol for $\mathcal{L}$ with the following property: there is a simulator pair* $\left(\mathsf{SLZK.Sim}_1,\ \mathsf{SLZK.Sim}_2^{\mathcal{H}_{\mathsf{tag}}}\right)$ *s.t. the protocol simulation is straight-line.*

Proof. **Completeness.** It follows by inspection. It is easy to see that if the original Σ-protocol $\Sigma.\Pi$ is perfectly complete, then the compiled SLZK protocol is also perfectly complete.

Soundness. The compiled SLZK protocol also preserves the special soundness property of the original Σ-protocol $\Sigma.\Pi$. First of all, the prover shows that either the statement $x \in \mathcal{L}$ or he/she knows α such that $f_{\mathsf{tag}_\ell}(\alpha) = \beta$. Since $\mathbf{F}$ is a publicly samplable adaptive one-way function family, the PPT adversary has negligible probability to find the pre-image α such that $f_{\mathsf{tag}}(\alpha) = \beta$, where $\beta \leftarrow \mathsf{IM}(\mathsf{tag}, C)$ and $C \in \{0,1\}^\lambda$ is randomly chosen by the verifier. Therefore,

```
1. $\mathsf{SLZK.Prv}_1(x, w)$:
   – Run $(a, \mathsf{st}) \leftarrow \Sigma.\mathsf{Prv}_1(x, w)$.
   – Pick random $R \leftarrow \{0,1\}^\lambda$.
   – Run $(a^*, \mathsf{st}^*) \leftarrow \mathsf{InD}.\Sigma.\mathsf{Sim}_1(R, \mathit{size})$, where $\mathit{size}$ is the circuit size of $f_{\mathsf{tag}}(\cdot)$.
   – Output $A := (a, a^*)$ and the state $\mathsf{st}_P := (R, \mathsf{st}, \mathsf{st}^*)$.
2. The verifier generates a random challenge $C \leftarrow \{0,1\}^\lambda$.
3. $\mathsf{SLZK.Prv}_2(x, w, C, \mathsf{st}_P)$:
   – Parse $\mathsf{st}_P = (R, \mathsf{st}, \mathsf{st}^*)$ and set $E := R \oplus C$.
   – Run $z \leftarrow \Sigma.\mathsf{Prv}_2(\mathsf{st}, E)$.
   – Run $\beta \leftarrow \mathsf{IM}(\mathsf{tag}, C)$ as in Def. 9 and define the statement as $x^* := \beta$ for $\mathcal{L}^*_{\mathsf{tag}}$
     as defined in Sec. 4.3, where $\mathsf{tag}$ is the verifier's PID.
   – Run $z^* \leftarrow \mathsf{InD}.\Sigma.\mathsf{Sim}_2(\mathsf{st}^*, x^*)$.
   – Output $Z := (E, z, z^*)$.
4. $\mathsf{SLZK.Verify}(x, A, C, Z)$:
   – Parse $A := (a, a^*)$ and $Z := (E, z, z^*)$ and compute $R := E \oplus C$.
   – Output 1 iff (i) $\Sigma.\mathsf{Verify}(x, a, E, z) = 1$ and $\mathsf{InD}.\Sigma.\mathsf{Verify}(x^*, a^*, R, z^*) = 1$.
```

Fig. 10: Three round straight-line simulatable ZK compiler.

the prover has to simulate the input-delayed proof for $\mathcal{L}^*_{\mathsf{tag}}$. That means the probability that a prover can produce an accepting tuple (x^*, a^*, R', z^*) where $R' \neq R$ is negligible. With overwhelming probability, after the prover outputs the first message A, given $C' \neq C$, the prover shall output a tuple (x, a, E', z') such that $E' = R \oplus C \neq E$. Hence, with two different challenges, we can obtain two accepting tuples (x, a, E, z) and (x, a, E', z') with overwhelming probability. By definition, there exist an extractor $\Sigma.\mathsf{Ext}$ that takes input as (x, a, E, z) and (x, a, E', z'), and outputs the witness w such that $(x, w) \in \mathcal{R}_{\mathcal{L}}$.

Straight-line simulatable zero-knowledge. The zero-knowledge property is based on the HVZK property of both the original Σ-protocol $\Sigma.\Pi$ and the input-delayed $\mathsf{InD}.\Sigma.\Pi$-protocol. The simulator consists of two algorithms $\big(\mathsf{SLZK.Sim}_1$, $\mathsf{SLZK.Sim}_2^{\mathcal{H}_{\mathsf{tag}}}\big)$ as follows.

- $\mathsf{SLZK.Sim}_1(x)$:
  - • Pick random $E \leftarrow \{0,1\}^\lambda$.
  - • Run $(a, E, z) \leftarrow \Sigma.\mathsf{Sim}(x, E)$.
  - • Run $(a^*, \mathsf{st}^*) \leftarrow \mathsf{InD}.\Sigma.\mathsf{Prv}_1(R, \mathit{size})$, where size is the circuit size of $f_{\mathsf{tag}}(\cdot)$.
  - • Output $A := (a, a^*)$ and the state $\mathsf{st}_S := ((E, z), \mathsf{st}^*)$.
- $\mathsf{SLZK.Sim}_2^{\mathcal{H}_{\mathsf{tag}}}(C, \mathsf{st}_S)$:
  - Parse $\mathsf{st}_S = ((E, z), \mathsf{st}^*)$ and set $R := E \oplus C$.
  - Run $\beta \leftarrow \mathsf{IM}(\mathsf{tag}, C)$ and define the statement as $x^* := \beta$ for $\mathcal{L}^*_{\mathsf{tag}}$.
  - • Query $(f_{\mathsf{tag}}, \beta)$ to the helper $\mathcal{H}_{\mathsf{tag}}$, and obtain α.
  - Run $z^* \leftarrow \mathsf{InD}.\Sigma.\mathsf{Prv}_2(\mathsf{st}^*, x^*, \alpha, R)$.
  - Output $Z := (E, z, z^*)$.

Due to the special HVZK property of $\Sigma.\Pi$, the simulated tuple (x, a, E, z) has identical distribution as the real one. Similarly, the tuple (x^*, a^*, R, z^*) is indistinguishable from the simulated one as in the real protocol transcript due to the

HVZK property of the $\mathsf{InD}.\Sigma$-protocol. Moreover, $\mathsf{SLZK}.\mathsf{Sim}_1$ does not require the challenge to produce the first message; therefore, the proposed protocol is straight-line simulatable zero-knowledge. □

F.5 Fully simulatable zero-knowledge in the $\mathcal{H}$-EUC model

The zero-knowledge functionality. In our construction, we also need the zero-knowledge functionality $\mathcal{F}_{\mathsf{ZK}}^{\mathcal{R}_{\mathcal{L}}}$ as decrypted in Fig. 11. In order to guarantee *privacy* when there is at least one honest server, for every ZK proof in VMPC, the server needs to prove it to all the rest servers. Note that this is important and should not be confused with *verifiability* when all the servers are corrupted. This is because in our construction, the CVZK proofs can only be verified at the end of the protocol. It is too late to prevent privacy leakage due to malicious execution. In addition, as the fully simulatable zero-knowledge proofs are between two servers, they are not restricted to 3 moves. There is a prover, S_i, $i \in [k]$, and several verifiers, $\{S_j\}_{j\in[k]\setminus\{i\}}$. The prover wants to convince the verifiers that a statement $x \in \mathcal{L}$, i.e. there exists $(w, x) \in \mathcal{R}_{\mathcal{L}}$. When there is at least one honest verifier, $\mathcal{F}_{\mathsf{ZK}}^{\mathcal{R}_{\mathcal{L}}}$ returns $(\textsc{Verify}, \mathsf{sid}, 1)$ if and only if $x \in \mathcal{L}$. When all the verifiers are corrupted, the output of $\mathcal{F}_{\mathsf{ZK}}^{\mathcal{R}_{\mathcal{L}}}$ can be decided by the adversary $\mathcal{A}$.

The zero-knowledge $\mathcal{F}_{\mathsf{ZK}}^{\mathcal{R}_{\mathcal{L}}}$ functionality

The functionality interacts with the prover P, the verifier V, and the adversary $\mathcal{A}$. It is parameterized by an **NP** relation $\mathcal{R}_{\mathcal{L}}$ for an **NP** language $\mathcal{L}$.

– Upon receiving $(\textsc{Statement}, \mathsf{sid}, x)$ from the verifier V and $(\textsc{Prove}, \mathsf{sid}, \langle x, w\rangle)$ the prover P:
  - Send $(\textsc{Leak}, \mathsf{sid}, x)$ to the adversary $\mathcal{A}$.
  - Set $b := 1$ if $(x, w) \in \mathcal{R}_{\mathcal{L}}$; otherwise, it sets $b := 0$.
  - It sends $(\textsc{Verified}, \mathsf{sid}, b)$ to V via public delayed output.

Fig. 11: The zero-knowledge functionality $\mathcal{F}_{\mathsf{ZK}}^{\mathcal{R}_{\mathcal{L}}}$.

$\mathcal{H}$-EUC Realization of $\mathcal{F}_{\mathsf{ZK}}^{\mathcal{R}_{\mathcal{L}}}$. To realize $\mathcal{F}_{\mathsf{ZK}}^{\mathcal{R}_{\mathcal{L}}}$ in the $\mathcal{H}$-EUC setting, we need to construct a straight-line extractable and simulatable zero-knowledge with access to the super-polynomial helper $\mathcal{H}$. The construction uses the perfectly binding commitment scheme under the adaptive DDH assumption as described in C.2. As depicted in Fig. 12, the prover needs to run the straight-line extractable and simulatable zero-knowledge with each of the verifiers. To enable straight-line extractability, we let the prover and the verifier first do a coin fliping protocol to jointly setup a public key pk. The prover then encrypts its witness w using pk and show that the encrypted w is a valid witness, i.e., $(x, w) \in \mathcal{R}_{\mathcal{L}}$ in zero-knowledge using SLZK as constructed in F.4.

Upon receiving ($\textsc{Statement}, \mathsf{sid}, x$) from the environment $\mathcal{Z}$, the verifier V does:

- Run $\mathsf{ck} \leftarrow \mathsf{CS}^{\mathbf{F}}.\mathsf{Gen}(1^\lambda)$ and $\mathsf{pk}_1 \leftarrow \mathsf{CS}^{\mathbf{F}}.\mathsf{Gen}(1^\lambda)$.
- Compute $e \leftarrow \mathsf{CS}^{\mathbf{F}}.\mathsf{Com}_{\mathsf{ck}}(\mathsf{pk}_1; r)$ with a fresh randomness r.
- Send (ck, e) to the prover P.

Upon receiving ($\textsc{Prove}, \mathsf{sid}, x, w$) from the environment $\mathcal{Z}$, the prover P does:

- Run $\mathsf{pk}_2 \leftarrow \mathsf{CS}^{\mathbf{F}}.\mathsf{Gen}(1^\lambda)$.
- Wait until it receives (ck, e) from the verifier V.
- Send pk_2 to the verifier V.

Upon receiving pk_2 from the prover P, the verifier V does:

- Run $\mathsf{pk}_2 \leftarrow \mathsf{CS}^{\mathbf{F}}.\mathsf{Gen}(1^\lambda)$.
- Wait until it receives (ck, e) from the verifier V.
- Set $\mathsf{pk} := \mathsf{pk}_1 \cdot \mathsf{pk}_2$.
- Send pk_1 to the verifier V.

Upon receiving pk_1 from the verifier V, the prover P does:

- Invoke a SLZK with V, letting V to prove there exists r s.t. $e = \mathsf{CS}^{\mathbf{F}}.\mathsf{Com}_{\mathsf{ck}}(\mathsf{pk}_1; r)$.
- Set $\mathsf{pk} := \mathsf{pk}_1 \cdot \mathsf{pk}_2$.
- Compute $c \leftarrow \mathsf{CS}.\mathsf{Com}_{\mathsf{pk}}(w; \rho)$ with a fresh randomness ρ.
- Send c to V.

Upon receiving c from the prover P, the verifier V does:

- Invoke a SLZK with P, letting P to prove there exists (w, ρ) s.t.

$$c \leftarrow \mathsf{CS}.\mathsf{Com}_{\mathsf{pk}}(w; \rho) \ \wedge\ (w, x) \in \mathcal{R}_{\mathcal{L}}$$

- If it verifies, return ($\textsc{Verified}, \mathsf{sid}, 1$) to the environment $\mathcal{Z}$; otherwise, return ($\textsc{Verified}, \mathsf{sid}, 0$) to $\mathcal{Z}$.

Fig. 12: Straight-line extractable and simulatable zero-knowledge $\mathit{\Pi}_{\mathsf{ZK}}^{\mathcal{R}_{\mathcal{L}}}$.

Theorem 9. *The protocol $\mathit{\Pi}_{\mathsf{ZK}}^{\mathcal{R}_{\mathcal{L}}}$ as described in Figure 12 $\mathcal{H}$-EUC-realizes $\mathcal{F}_{\mathsf{ZK}}^{\mathcal{R}_{\mathcal{L}}}$ against static corruption.*

F.6 Realization of $\mathcal{F}_{\mathsf{V.Offline}}$ in the $\mathcal{H}$-EUC model

Many works, e.g., [12, 57, 29, 28, 4] in the literature provide various MPC realizations of the SPDZ-type of offline functionality. However our $\mathcal{F}_{\mathsf{V.Offline}}$ functionality is slightly different, as it *needs* to be crowd verifiable. Moreover, we would like to realized it in the $\mathcal{H}$-EUC model without trusted setups. It is shown in [58, 18, 17] that any *well-defined* functionality can be realized in the $\mathcal{H}$-EUC model. Such a protocol can guarantee the correctness of the output as long as at least one party remain honest. In addition, to achieve crowd verifiability when all the servers are corrupted, we need to transform all the ZK proofs in the MPC protocol to CVZK proofs. In the following, we provide one possible realization of the $\mathcal{F}_{\mathsf{V.Offline}}$ functionality in the $\mathcal{H}$-EUC model.

Fig. 14 depicts a more efficient realization of $\mathcal{F}_{\mathsf{V.Offline}}$ in the $\mathcal{H}$-EUC model. For readability, we assume there exists a protocol that secure realize $\mathcal{F}_{\times}$ in the $\mathcal{H}$-EUC mode; for instance, using Gilboa's multiplication [34] as in [42] to compute two party private multiplication in the $\mathcal{F}_{\mathsf{OT}}$-hybrid world. The two-party multiplication functionality $\mathcal{F}_{\times}$ is illustrated in Fig. 13.

The two-party multiplication functionality $\mathcal{F}_{\times}$

The functionality interacts with two parties P_1, P_2 and the adversary $\mathcal{A}$.

– Upon receiving $(\textsc{Multiply}, \mathsf{sid}, x, \mathbb{F})$ from P_1 and $(\textsc{Multiply}, \mathsf{sid}, y, \mathbb{F})$, it computes $z = x \times y$, where $\times$ is the field operation according to $\mathbb{F}$. It then sends $(\textsc{Output}, \mathsf{sid}, z)$ to all the parties P_1 and P_2 via private delayed channel.

Fig. 13: The two-party multiplication functionality $\mathcal{F}_{\times}$

First of all, note that we use the user's coin $b_\ell \in \{0, 1\}$ as the verifier's challenge in the CVZK. Namely, the prover first runs $\mathsf{CVZK.Prv}_1$ in the preparation phase. Then the users post their coins on the $\mathcal{G}_{\text{BB}}$ during the input phase. Finally, the prover finishes the proof using $\mathsf{CVZK.Prv}_2$ in the computing phase.

As such, CVZK has delayed confirmation, i.e., the result will be known after the computing phase. It is sufficient for correctness, but it is not enough to achieve the level of privacy we want. This is because we want to guarantee the user's input privacy as far as there is one honest server. However, all the CVZK proofs are not verifiable during the input phase. Any malicious server can cause privacy leakage by deviating from the protocol. Of course, this malicious behavior can be detected in the computing phase from CVZK. However, the user's privacy is already leaked in the input phase.

To address this issue, all the zero-knowledge proofs must be proven to all the servers using either SLZK or $\mathcal{F}_{\mathsf{ZK}}$. Therefore, if at least one server is honest, the VMPC correctness can be ensured by SLZK or $\mathcal{F}_{\mathsf{ZK}}$; if all the servers are corrupted, the VMPC correctness can be ensured by CVZK.

In **Phase 1**, each of the servers $S_i \in \mathcal{S}$ generates a partial ElGamal public key $\mathsf{pk}_i \leftarrow \mathsf{EG.Gen}(\mathsf{Param}_{\mathsf{gp}})$. It then commits pk_i using a perfectly binding commitment. After posting the commitment of pk_i on the $\mathcal{G}_{\text{BB}}$, S_i prove the knowledge of pk_i using $\mathcal{F}_{\mathsf{ZK}}$ to all the other servers $S_j \in \mathcal{S} \setminus \{S_i\}$.

In **Phase 2**, each server $S_i \in \mathcal{S}$ posts pk_i to the $\mathcal{G}_{\text{BB}}$ and proves the opening is correct using SLZK. This would allow the simulator to equivocate the opening in our proof.

The offline protocol $\Pi_{\mathsf{offline}}^{\mathcal{F}_{\times},\mathcal{F}_{\mathsf{ZK}},\mathcal{G}_{\mathrm{BB}}}$

– Upon receiving $(\textsc{Init}, \mathsf{sid}, n_1, n_2)$ from the environment, the server $S_i \in \mathcal{S}$ does the following.

Phase 1:

- Run $\mathsf{ck}_i \leftarrow \mathsf{EG.Gen}(\mathsf{Param_{gp}})$ and $\mathsf{pk}_i \leftarrow \mathsf{EG.Gen}(\mathsf{Param_{gp}})$.
- Compute $e_i \leftarrow \mathsf{EG.Com}_{\mathsf{ck}_i}(\mathsf{pk}_i; r_i)$ with a fresh randomness $r_i \leftarrow \mathbb{Z}_p$.
- Post e_i (i.e. send $(\textsc{write}, \mathsf{sid}, \mathsf{pid}, x)$) to $\mathcal{G}_{\mathrm{BB}}$ and prove the knowledge of (pk_i, r_i) using $\mathcal{F}_{\mathsf{ZK}}$.

Phase 2:

- Post pk_i to $\mathcal{G}_{\mathrm{BB}}$, and show that there exists $r_i \in \mathbb{Z}_p$ s.t. $e_i = \mathsf{EG.Com}_{\mathsf{ck}_i}(\mathsf{pk}_i; r_i)$ using SLZK.

Phase 3:

- Set $\mathsf{pk} = \prod_{i=1}^{k} \mathsf{pk}_i$. Compute $u_i \leftarrow \mathsf{Enc}_{\mathsf{pk}}(0; t_i)$ with fresh randomness $t_i \leftarrow \mathbb{Z}_p$.
- Post u_i to $\mathcal{G}_{\mathrm{BB}}$ and show that there exists $t_i \in \mathbb{Z}_p$ s.t. $u_i = \mathsf{Enc}_{\mathsf{pk}}(0; t_i)$ using both SLZK and CVZK.

Phase 4:

- Set $u = \mathsf{Enc}_{\mathsf{pk}}(1; 0) \cdot \prod_{i=1}^{k} u_i$. Define $\mathsf{ck} := (\mathsf{pk}, u)$.
- For $j \in [n_1]$, pick random $(r_{i,j}, \rho_{i,j}) \leftarrow (\mathbb{Z}_p)^2$; compute and post $R_{i,j} \leftarrow \mathsf{DC.Com}_{\mathsf{ck}}(r_{i,j}, \rho_{i,j})$ to $\mathcal{G}_{\mathrm{BB}}$; prove the knowledge of $(r_{i,j}, \rho_{i,j})$ using $\mathcal{F}_{\mathsf{ZK}}$ and CVZK.
- For $j \in [n_2]$, pick random $(a_{i,j}, \alpha_{i,j}, b_{i,j}, \beta_{i,j}) \leftarrow (\mathbb{Z}_p)^4$; compute and post $A_{i,j} \leftarrow \mathsf{DC.Com}_{\mathsf{ck}}(a_{i,j}, \alpha_{i,j})$ and $B_{i,j} \leftarrow \mathsf{DC.Com}_{\mathsf{ck}}(a_{i,j}, \beta_{i,j})$ to $\mathcal{G}_{\mathrm{BB}}$; prove the knowledge of $(a_{i,j}, \alpha_{i,j}, b_{i,j}, \beta_{i,j})$ using $\mathcal{F}_{\mathsf{ZK}}$ and CVZK.

Phase 5:

- For $j \in [n_1]$, set $R_j = \prod_{i=1}^{k} R_{i,j}$.
- For $j \in [n_2]$, for all the $x, y \in [k]$, do the following:
  - ∗ If $x = y$, do the following step locally.
  - ∗ S_x and S_y invoke $\mathcal{F}_{\times}$ to obtain $t_{x,y,j}^{(x)}$ (for S_x) and $t_{x,y,j}^{(y)}$ (for S_y) s.t. $t_{x,y,j}^{(x)} + t_{x,y,j}^{(y)} = a_{x,j} \cdot \beta_{y,j}$.
  - ∗ S_x computes and posts $C_{x,y,j} = B_{y,j}^{a_{x,j}} \cdot \mathsf{DC.Com}_{\mathsf{ck}}(0; t_{x,y,j}^{(x)})$ to $\mathcal{G}_{\mathrm{BB}}$. It then show the correctness of $C_{x,y,j}$ using CVZK.
  - ∗ S_y computes and posts $C'_{x,y,j} \leftarrow \mathsf{DC.Com}_{\mathsf{ck}}(0; t_{x,y,j}^{(y)})$. It then show that $C'_{x,y,j}$ is indeed a commitment of 0 using CVZK.
- For $j \in [n_2]$, set the Beaver triples as $A_j = \prod_{i=1}^{k} A_{i,j}$, $B_j = \prod_{i=1}^{k} B_{i,j}$, and $C_j = \prod_{x=1,y=1}^{k,k} (C_{x,y,j} \cdot C'_{x,y,j})$.

–

Fig. 14: The offline protocol $\Pi_{\mathsf{offline}}^{\mathcal{F}_{\times},\mathcal{F}_{\mathsf{ZK}},\mathcal{G}_{\mathrm{BB}}}$ in the $\{\mathcal{F}_{\times}, \mathcal{F}_{\mathsf{ZK}}, \mathcal{G}_{\mathrm{BB}}\}$-hybrid model.

In **Phase 3**, each server $S_i \in \mathcal{S}$ sets $\mathsf{pk} = \prod_{i=1}^{k} \mathsf{pk}_i$ and produces a ciphertext $u_i \leftarrow \mathsf{Enc}_{\mathsf{pk}}(0; t_i)$. It then shows the knowledge of t_i with respect to u_i using both SLZK and CVZK.

In **Phase 4**, each server $S_i \in \mathcal{S}$ defines the dual-mode commitment key as $u = \mathsf{Enc}_{\mathsf{pk}}(1; 0) \cdot \prod_{i=1}^{k} u_i$ and $\mathsf{ck} := (\mathsf{pk}, u)$. Note that ck is a perfectly binding key. Each server $S_i \in \mathcal{S}$ then generates commitments $\{R_{i,j}\}_{j\in[n_1]}$, $\{A_{i,j}\}_{j\in[n_2]}$ and $\{B_{i,j}\}_{j\in[n_2]}$, and proves the correctness of those commitments using $\mathcal{F}_{\mathsf{ZK}}$ and CVZK.

In **Phase 5**, we define $R_j = \prod_{i=1}^{k} R_{i,j}$, $j \in [n_1]$; $A_j = \prod_{i=1}^{k} A_{i,j}$, $j \in [n_2]$ and $B_j = \prod_{i=1}^{k} B_{i,j}$, $j \in [n_2]$. The servers use $\mathcal{F}_{\times}$ to compute all the cross terms to produce C_j such that (A_j, B_j, C_j) are commitments of a Beaver triple.

Theorem 10. *The protocol $\Pi_{\mathsf{offline}}^{\mathcal{F}_{\times}, \mathcal{F}_{\mathsf{ZK}}, \mathcal{G}_{\mathrm{BB}}}$ described in Fig. 14 $\mathcal{H}$-EUC realizes $\mathcal{F}_{\mathsf{V.Offline}}$ in the $\{\mathcal{F}_{\times}, \mathcal{F}_{\mathsf{ZK}}, \mathcal{G}_{\mathrm{BB}}\}$-hybrid model.*

Proof. (Sketch) To prove the theorem, we construct a simulator Sim such that no non-uniform PPT environment $\mathcal{Z}$ can distinguish between (i) the real execution in the $\{\mathcal{F}_{\times}, \mathcal{F}_{\mathsf{ZK}}, \mathcal{G}_{\mathrm{BB}}\}$-hybrid world and the corrupted parties are controlled by a dummy adversary $\mathcal{A}$ who simply forwards messages from/to $\mathcal{Z}$, and (ii) the ideal execution where the parties interact with functionality $\mathcal{F}_{\mathsf{V.Offline}}$ in the $\mathcal{G}_{\mathrm{BB}}$-hybrid model and corrupted parties are controlled by the simulator Sim. Moreover, there is a super-polynomial helper $\mathcal{H}$ that exists in both ideal and real world. Let $\mathcal{S}^*$ be the set of corrupted MPC servers. We consider the following cases.

Case 1: $0 \le |\mathcal{S}^*| < k$ (i.e. there is at least one honest server)

Simulator. The simulator Sim internally runs $\mathcal{A}$, forwarding messages to/from the environment $\mathcal{Z}$. The simulator Sim simulates honest server $S_i \in \mathcal{S} \setminus \mathcal{S}^*$ and functionalities $\mathcal{F}_{\times}, \mathcal{F}_{\mathsf{ZK}}$ and $\mathcal{G}_{\mathrm{BB}}$. In addition, the simulator Sim simulates the following interactions with $\mathcal{A}$.

- In **Phase 1**, the simulator Sim extracts the committed pk_i for all the corrupted servers $S_i \in \mathcal{S}^*$ by checking the internal state of the simulated $\mathcal{F}_{\mathsf{ZK}}$. The simulator sends $(\textsc{Init}, \mathsf{sid}, n_1, n_2)$ to $\mathcal{F}_{\mathsf{V.Offline}}$ on behave of all the corrupted servers $S_i \in \mathcal{S}^*$.
- In **Phase 2**, upon receiving $(\mathrm{CK}, \mathsf{sid}, \mathsf{ck})$ from $\mathcal{F}_{\mathsf{V.Offline}}$, the simulator Sim parses $\mathsf{ck} = (\mathsf{pk}^*, u^*)$. Define $\mathsf{pk}_i := \mathsf{pk}^* / \prod_{j\in[k], j\neq i} \mathsf{pk}_j$. It then posts pk_i to $\mathcal{G}_{\mathrm{BB}}$ and simulates the corresponding SLZK proof.
- In **Phase 3**, the simulator Sim acts as one of the honest servers $S_i \in \mathcal{S} \setminus \mathcal{S}^*$; it generates $u_i = u^* / (\mathsf{Enc}_{\mathsf{pk}}(1;0) \cdot \prod_{j=1, j\neq i}^{k} u_j)$ and simulates the corresponding SLZK and CVZK proofs.
- In **Phase 4** and **Phase 5**, the simulator receives $(\textsc{RandShare}, \mathsf{sid}, (r_{i,j}, \rho_{i,j})_{j\in[n_1]})$ and $(\textsc{TripleShare}, , \mathsf{sid}, (a_{i,j}, \alpha_{i,j}, b_{i,j}, \beta_{i,j}, c_{i,j}, \gamma_{i,j})_{j\in[n_2]})$ from $\mathcal{F}_{\mathsf{V.Offline}}$. It then fakes the corresponding shares and commitments accordingly.

Indistinguishability. It is straightforward, as the SLZK, CVZK proofs are simulatable in $\mathcal{H}$-EUC model, and $\mathcal{F}_{\mathsf{ZK}}$ is extractable and simulatable.

Case 2: $|\mathcal{S}^*| = k$

Simulator. Trivial case. There is nothing needs to extract, as the servers do not have input. The simulator Sim just runs the corrupted servers internally and submit to $\mathcal{F}_{\mathsf{V.Offline}}$ accordingly.

Indistinguishability. The indistinguishability in this case is straightforward, as Sim never simulate a single message as all the servers are corrupted. It is easy to see that the view of $\mathcal{Z}$ in the ideal execution has identical distribution to the view of of $\mathcal{Z}$ in the real execution. $\square$